VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZAZNAMENÁVÁNÍ TRASY JÍZDY NA MOTORCE PRO IOS

TRACKER OF MOTORCYCLE TRIPS FOR IOS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARTIN PINKA**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Pinka Martin**
Obor: Informační technologie
Téma: **Zaznamenávání trasy jízdy na motorce pro iOS**
Tracker of Motorcycle Trips for iOS
Kategorie: Uživatelská rozhraní

Pokyny:

1. Seznamte se s problematikou tvorby aplikací pro iOS.
2. Vyhledejte a popište existující nástroje pro záznam tras (cyklo, moto, auto, ...).
3. Navrhněte funkčnost aplikace pro záznam tras a dalších parametrů jízdy na motocyklu.
4. Prototypujte dílčí prvky uživatelského rozhraní a funkčnosti řešené aplikace. Testujte je na uživatelích.
5. Vytvořte řešenou aplikaci, testujte ji na uživatelích a iterativně zdokonalujte.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- dle pokynů vedoucího

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Herout Adam, prof. Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cílem této práce je vytvořit mobilní aplikaci pro zaznamenávání jízdy na motocyklu na platformě iOS, která inovuje aktuálně dostupné řešení. Mezi přínosy patří vytvoření lepšího mechanismu pro detekci přestávky zaznamenávání, který umožní automatizovat pozastavování zaznamenání trasy v době, kdy uživatel nevykonává jízdu na motocyklu. Dále je aplikace rozšířena o organizaci záznamů a jejich vyhledávání. Ta je řešena pomocí tagů, které lze k záznamům přidat. Řešení významně zlepšuje rychlost vyhledání konkrétních jízd, pokud uživatel poctivě vyplní informace k jízdě. V implementační části práce je popsáno vykreslování velmi dlouhé trasy na mapě, jejíž části jsou barevně odlišeny podle rychlosti, která byla při zaznamenávání zaznamenána. Výsledkem celého úsilí je inovativní funkční aplikace, která je připravena pro reálný provoz.

Abstract

The aim of the thesis is to create an innovative mobile application that can be used for tracking of motorcycle trips on iOS platform. The benefits include the improvement of automatic pause detection mechanism, which ignores movements that were not done by riding a motobike. These parts are not included in the trip. Additionally, the application provides a way to organize records which helps user to find specific recorded trip. This is solved by using tags that can be added to the records. The solution greatly improves the speed of finding specific routes if the user honestly fills information about trips. The implementation part of the thesis describes the drawing of a very long colored polyline on the map where colors represent speed of passing the places. As a result of all the effort there is an innovative, functional application that is ready for real usage.

Klíčová slova

mobilní aplikace, iPhone, iOS, MapKit, GPS, Swift, zaznamenávání trasy

Keywords

mobile application, iPhone, iOS, MapKit, GPS, Swift, route tracking

Citace

PINKA, Martin. *Zaznamenávání trasy jízdy na motorce pro iOS*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Herout Adam.

Zaznamenávání trasy jízdy na motorce pro iOS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana profesora Adama Herouta. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Pinka
16. května 2017

Poděkování

Velmi děkuji svému vedoucímu práce, profesoru Adamu Heroutovi, za vynikající odbornou pomoc, věcné připomínky a nápady, které jsem do práce zahrnul.

Obsah

Kapitola 1

Úvod

Práce se zabývá vývojem aplikace pro zaznamenávání trasy na platformě iOS. Při mých pravidelných vyjížďkách na motocyklu, které nebyly dopředu plánovány, jsem se držel hesla „cesta je cíl“. Po návratu domů jsem chtěl vědět, kudy jsem vlastně jel. Z tohoto důvodu jsem začal zkoušet aplikace, které sloužily pro zaznamenávání trasy. Nejprve se jednalo o aplikace určené k zaznamenávání sportovních aktivit, o kterých jsem věděl, že fungují a dokáží přesně zaznamenat trasu. Ty však pro můj účel nebyly úplně vhodné, neboť obsahovaly informace, které nebyly pro jízdu na motocyklu relevantní a uživatelské rozhraní nebylo pro toto používání přizpůsobeno. Pak jsem objevil aplikace určené přímo pro milovníky jednostopých vozidel. Některé jsem si nainstaloval. Ani ty však úplně nevyhovovaly mým požadavkům. Proto jsem se rozhodl vytvořit vlastní řešení.

Práce shrnuje nedostatky existujících programů. Na jejich základě je v textu popsán návrh aplikace, která je inovativní zejména díky organizaci jízd pomocí tagů. Ty značně zjednodušují vyhledávání záznamů.

V textu je zahrnuta problematika vývoje aplikací pro platformu iOS. Dále je popsána implementace samotné aplikace, která byla vytvořena podle vypracovaného návrhu řešení. Zaměřil jsem se hlavně na vykreslování ujeté trasy na mapě s barevně znázorněnou rychlostí pro velmi dlouhé záznamy, dále mechanismus automatické pauzy během zaznamenávání a grafu dob strávených v rychlostech. K jeho zobrazení byla vytvořena znovupoužitelná komponenta.

Kapitola 2

Konkurenční aplikace pro záznam trasy

Tato kapitola popisuje výhody a nevýhody konkurenčních aplikací, které slouží k zaznamenávání trasy. Je zaměřena na celkem čtyři z těchto aplikací. Ty byly vybrány na základě průzkumu v motorkářské komunitě, jejíž členové byli dotázáni, jakou mobilní aplikaci používají pro záznam jízdy na motocyklu. Nejčastěji zmiňovanou byla aplikace pro motorkáře primárně vytvořená „BikerSeason“. Mezi odpovědmi se velmi často vyskytovaly i aplikace určené pro zaznamenávání pohybových aktivit. V této kategorii se jednalo především o aplikace „Runtastic“. Dále byla do přehledu zařazena v České republice téměř neznámá aplikace „Rever“, neboť tento americký projekt uzavřel spolupráci s BMW Motorrad [5].

2.1 Runtastic

Pod značkou „Runtastic“ na Apple Store naleznete více produktů pro vedení historie fyzických aktivit. Dvěma nejstahovanějšími, které zaznamenávají trasu podle GPS, jsou „Runtastic GPS Running, Jogging and Fitness Tracker“ (dále jen Running Tracker) a „Runtastic Mountain Bike & Route Tracker“ (dále jen Route Tracker). Obě aplikace jsou dostupné nejen v placené verzi, ale i zdarma. Mají velmi mnoho společného, ať už se jedná o pohled vizuální nebo funkční.

Jak již názvy napovídají, první jmenovaná aplikace je určena primárně pro běhání, druhá je zaměřena na cyklistiku. Protože cyklistická aplikace neobsahuje řadu funkcí, jako je například ovládání přehrávané hudby během puštěného zaznamenávání nebo tréninkové plány, dá se říci, že se jedná o zjednodušenou verzi běžecké aplikace. Dalším rozdílem je, že v historii Running Trackeru lze nalézt i aktivity z dalších aplikací od „Runtastic“. Ty se však dají filtrovat. Kromě samotných aplikací je možné si vlastní záznamy prohlédnout na webových stránkách. Z tohoto důvodu je při prvním spuštění vyžádána registrace uživatele.

Když se zaměříme na detail aktivity v Running Trackeru (obr. 2.1), vidíme vykreslenou trasu na mapě. Na rozdíl od Route Trackeru se již v základní verzi vykresluje trasa barevně v závislosti na rychlosti uživatele v daném místě. Dále na obrazovce nalezneme informace o průměrném tempu, rychlosti, maximální rychlosti nebo o nastoupané či sestoupené nadmořské výšce.

Společnou funkcí této dvojice je automatická pauza, kterou lze u pohybových aktivit poměrně snadno naimplementovat pomocí knihovny CoreMotion, neboť již obsahuje způsob detekce chůze, běhu, jízdy na kole nebo zda se uživatel nachází ve vozidle.

Obrázek 2.1: Aplikace Runtastic – Detail uložené aktivity

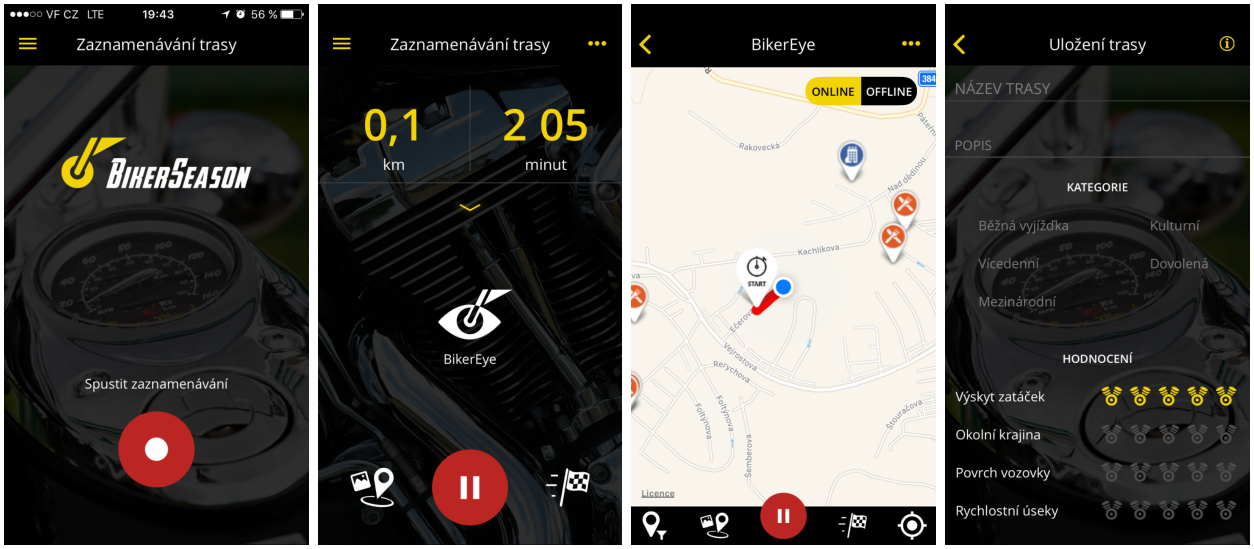
2.2 BikerSeason

Jedná se o českou aplikaci cílenou na řidiče motocyklů. Podobně jako předchozí zmíněné řešení vyžaduje registraci, a to pomocí sociální sítě Facebook nebo pomocí e - mailu. Během registrace je nutno vyplnit i značku a typ motocyklu. Je však možné zvolit jen jeden motocykl. Na majitele více strojů vývojáři nemysleli.

Po přihlášení se zobrazí jednoduchá obrazovka (obr. 2.2 – Spuštění zaznamenávání) s jedinou funkcí, kterou je spuštění zaznamenávání. Jakmile ho spustíme, uvidíme údaje o ujeté vzdálenosti a stráveném čase, jak je zřejmé na obrázku 2.2 – průběh zaznamenání. Tlačítko s popisem „BikerEye“ přesměruje uživatele na mapu s vyznačenými body zájmu a ujetou trasou (obr. 2.2 zaznamenaná trasa). Mezi další akce, které je možné vykonat během záznamu, patří pořízení fotografie, pozastavení a dokončení zaznamenání. Všechny tyto funkce jsou dostupné i přes nabídku „více“, která je symbolizována třemi tečkami v pravém horním rohu obrazovky.

Pro uložení záznamu (obr. 2.2 uložení záznamu) je nutné vyplnit název, vybrat alespoň jednu z 5 dostupných kategorií a ohodnotit alespoň jednu hodnotící kategorii. Nutnost vyplňovat tolik informací pro uložení může být pro uživatele nepříjemně zdlouhavá. Dále je zde možnost trasu označit jako soukromou nebo veřejnou. V druhém případě ji pak ostatní uživatelé budou moci vidět v seznamu tras (obrázek 2.3 vlevo). Po kliknutí na tlačítko „uložit trasu“ jste dotázáni, zda chcete jízdu nahrát na BikerSeason jen při připojení na WiFi nebo i přes mobilní data. Do momentu, kdy se jízda nahraje na server, je dostupná i bez připojení k internetu. Poté ji můžeme zpřístupnit v režimu offline jen tehdy, pokud si zaplatíme premium účet. To se týká i tras naplánovaných pomocí webového rozhraní i vytvořených jinými uživateli. Premium učet dále obohatí aplikaci o možnost plánování tras, export trasy do KML nebo GPX formátu, nebo plánování výletů.

Na obrázku 2.3 vlevo je obrazovka, na které se zobrazují plánované i zaznamenané trasy. Ty však nejsou nijak odlišené, což lze považovat za velmi matoucí. Dále si v seznamu lze zobrazit nejbližší trasy, nejnovější trasy či jejich žebříček, a to pomocí filtru vyobrazeném



Obrázek 2.2: Aplikace BikerSeason – zleva: Spuštění zaznamenávání, průběh zaznamenaní, zaznamenaná trasa, uložení záznamu

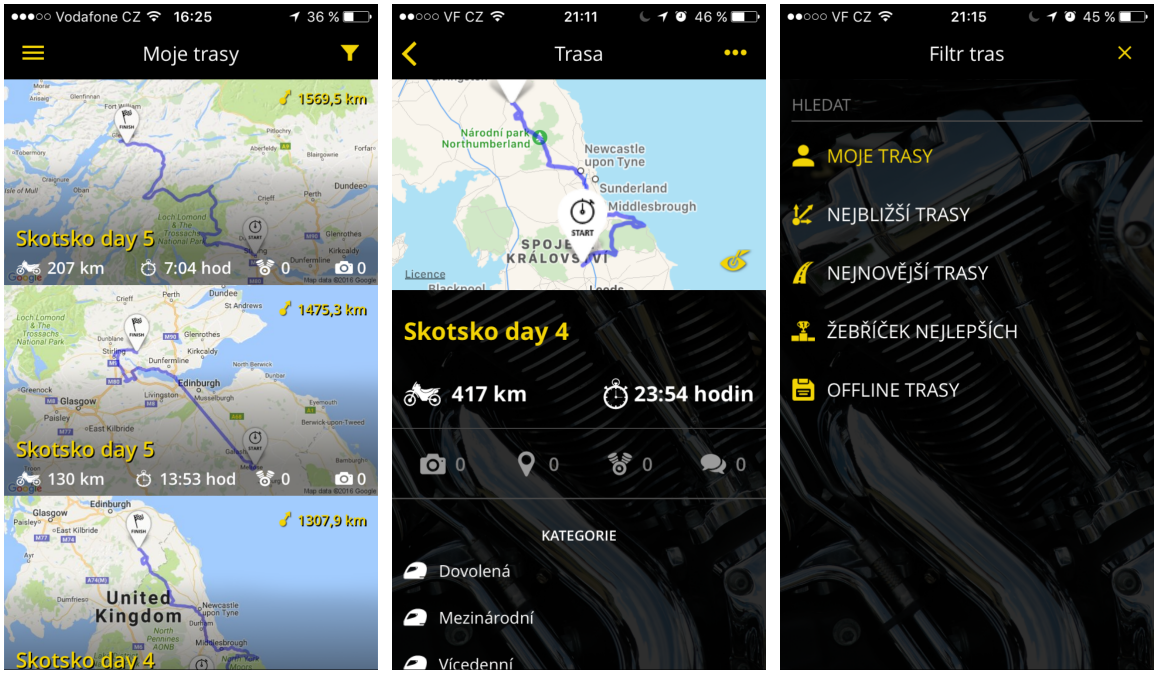
na obrázku 2.3 vpravo. Nalezneme zde i vyhledání podle jména. Výsledkem jsou jen trasy všech uživatelů, kteří je označili jako veřejné. Pokud klikneme na nějakou trasu ze seznamu, dostaneme se na její detail (obr. 2.3 uprostřed). Tam nalezneme informace o vzdálenosti, stráveném času, kategoriích, které uživatel k trase vyplnil a bodech zájmu. V detailu, stejně jako v položkách v seznamu, nejsou žádné odlišnosti mezi naplánovanou a zaznamenanou trasou. V případě, že se jedná o záznam, pak údaj o čase označuje skutečně strávený čas na cestě. V případě naplánované trasy zobrazuje dobu, za kterou je možné trasu ujet.

2.3 Rever

Rever je další aplikace určená pro záznam trasy na motocyklu. Rovněž si po spuštění vyžádá registraci nebo přihlášení uživatele. Po přihlášení je možné začít okamžitě nahrávat jízdu, jak je zřejmé na obrázku 2.4 vlevo. Aktuálně zaznamenávanou trasu (obr. 2.4 uprostřed) můžeme vidět po kliknutí na záložku „map“.

V aplikaci, podle obrázku 2.4 vpravo, nalezneme možnost zapnutí automatické pauzy. Při jejím testování spočívajícím v necelých 3 hodinách zaznamenávání, které zahrnovalo cestu do garáže, přípravu motocyklu a následnou dvou a půl hodinovou jízdu s několika přestávkami, byl naměřen záznam dlouhý 2 hodiny 55 minut. Naměřená kilometrová vzdálenost byla oproti tachometru na motocyklu větší o 700 metrů. Tento rozdíl by mohl být způsoben samotným tachometrem motocyklu, který má obvykle odchylku závislou na měnícím se obvodu kola (sjíždění dezénu) nebo na použití jiného rozměru kola, než pro který byl tachometr nakonfigurován. Na obrázcích 2.5 však vidíme záznam pohybu v době, kdy motocykl nejel, ale uživatel chodil. Při dalším testování bylo zjištěno, že se pauza nepravidelně aktivuje v době pohybu v budovách, což však není dostatečné.

Uživatel si může předplatit premium účet, který mu umožní notifikovat přátele při spuštění a dokončení zaznamenávání nebo je sledovat během jejich jízdy. Dále pak stejně jako v BikerSeason je umožněno stažení map do telefonu.



Obrázek 2.3: Aplikace BikerSeason. **vlevo:** Seznam tras, **uprostřed:** Detail trasy, **vpravo:** Filtrování a hledání trasy

2.4 Podpora Apple Watch

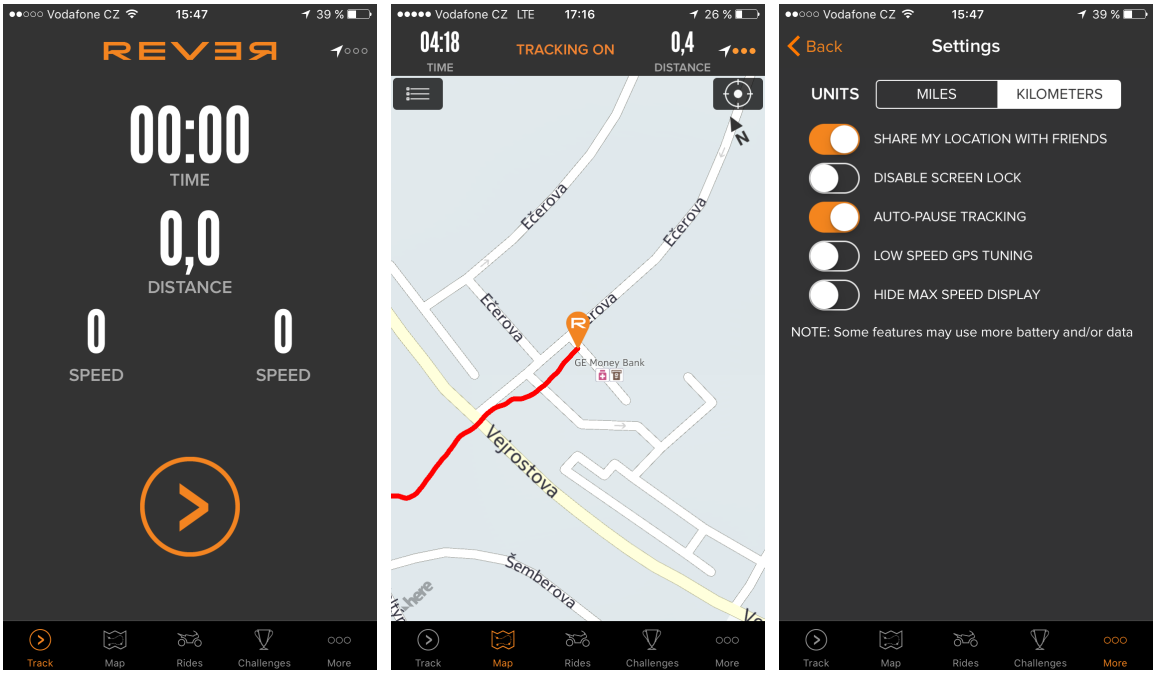
Dvě z vyjmenovaných aplikací nabízí podporu pro Apple Watch. Vzhledem k tomu, že se jedná o aplikace specializované na sportovní aktivity, při kterých není nutné používat výstroj zamezující přístupu k předloktí, je tato podpora zcela logická. U aplikace orientující se na řidiče motocyklů, však toto neplatí. Lze předpokládat, že většina jezdců již používá adekvátní a bezpečnou výstroj, která ztěžuje přístup k hodinkám na předloktí. Ta je např. v Belgii uzákoněna jako povinná [11]. Díky znalosti této skutečnosti lze tvrdit, že rozšíření aplikace na Apple Watch je pro řidiče motocyklů zbytečné.

2.5 Shrnutí nedostatků a návrhy pro vylepšení

Za hlavní problém mezi aplikacemi určenými pro zaznamenávání jízd patří registrace a nutnost přihlášení. Jakmile by se uživatel omylem odhlásil bez přístupu k internetu, nebo by nastala chyba v podobě ztráty „user session“, není možné dále zaznamenávat trasu. To při cestování nejen v zahraničí může způsobit nechtěné problémy s aplikací.

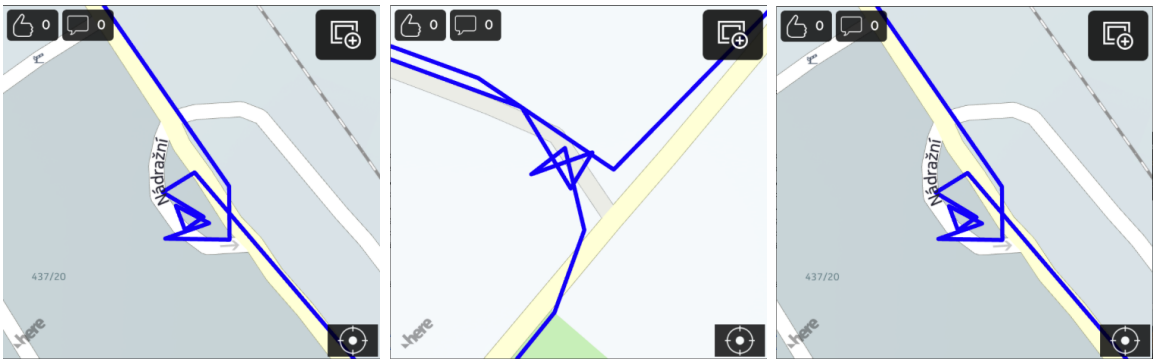
Kromě aplikace BikerSeason všechny obsahovaly možnost automatické pauzy s různými způsoby implementace. Ty ve sportovních řešeních však nejsou použitelné na motocyklu, neboť pozastavení resp. spuštění je detekováno na základě pohybů zařízením. Automatická pauza v aplikaci Rever čelí nedostatkům při pohybu mimo budovy a proto je zahrnuta v této práci.

V případě dlouhodobého používání se v žádném z řešeních nevyskytuje způsob, jak dohledat konkrétní trasu, kromě neefektivního procházení celého seznamu. Pro snadné dohledání by bylo vhodné k jízdám přiřazovat štítky neboli tagy, které by ji blíže charakterizovaly (např. Slovensko, off-road, slunečno). Ty navíc pomohou k organizaci jízd v případě více-



Obrázek 2.4: Aplikace Rever pro zaznamenávání tras. **vlevo:** Spuštění zaznamenávání, **uprostřed:** Zaznamenaná trasa, **vpravo:** Nastavení

denních cest. Dále je zde možnost aplikace vylepšit z hlediska rozšíření statistik záznamů. V žádné aplikaci se např. nevyskytly informace, jakou dobu uživatel strávil v určité rychlosti. V případě celkových statistik chybí grafy nájezdu kilometrů v jednotlivých dnech či měsících nebo informace o průměrném denním nájezdu.



Obrázek 2.5: Aplikace rever – mapa v detail záznamu: místa, kde nebyl motocykl v pohybu ale i tak, byla zaznamenána změna lokace telefonu

Kapitola 3

Vývoj aplikací pro iOS

Tato kapitola popisuje dostupné prostředky k vývoji aplikací pro iOS. Popisuje vývojové prostředí, možnost návrhu uživatelského rozhraní a nástroje k odhalování chyb. Dále popisuje nástroje pro správu knihoven třetích stran.

3.1 Vývojové prostředí

Prakticky jediným používaným vývojovým prostředí pro tvorbu aplikací pro platformu iOS je Xcode. Tento nástroj obsahuje vše potřebné, co vývojář potřebuje k vývoji aplikací na danou platformu od nástrojů pro debuggování, testování po správu certifikátů či odeslání aplikace do iTunes.

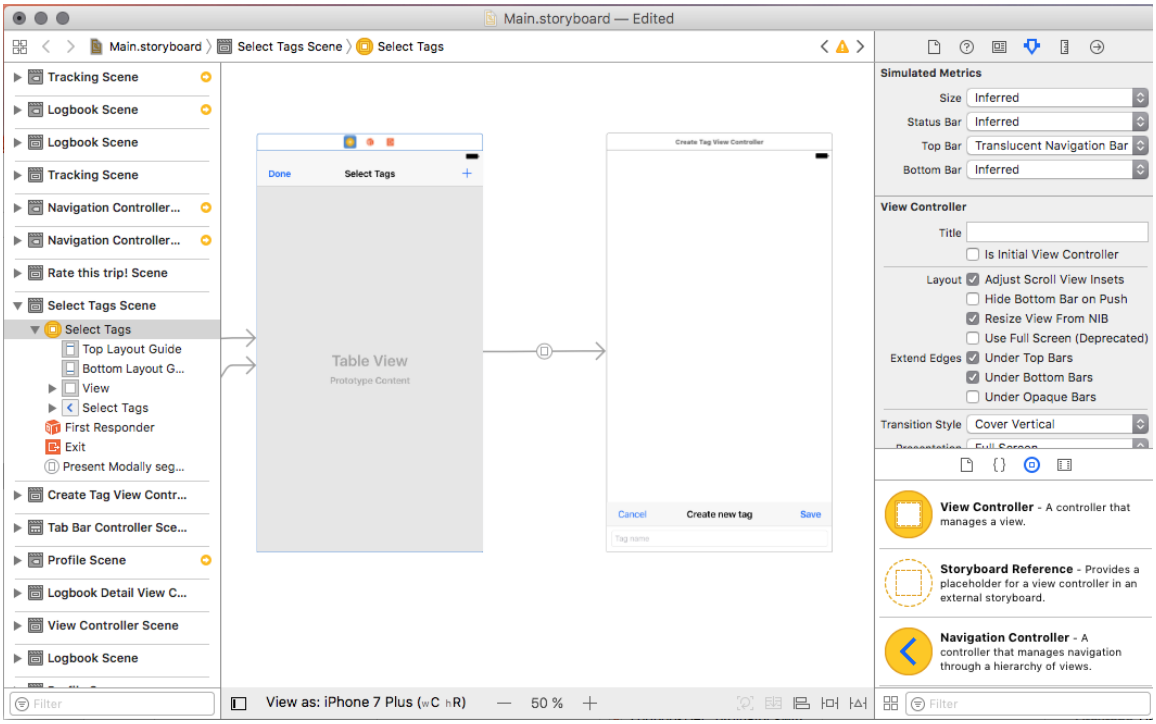
3.1.1 Instruments

Instruments je nástroj pro analýzu a testování výkonu, který je součástí sady nástrojů v Xcode. Pomáhá k analýze chování aplikací k optimalizaci využití zdrojů. Na rozdíl od jiných nástrojů pro odhalování chyb umožňuje shromažďovat velmi rozmanité typy dat a zobrazovat je vedle sebe[1]. To usnadňuje nalezení zdroje problému např. při přílišném vytížení procesoru nebo využití paměti.

Na obrázku 3.2 je diagram pracovního postupu pří používání Instruments. Ten však není nutné používat pravidelně, neboť při každém spuštění aplikace přes Xcode je dostupný graf využívaných zdrojů. Tento postup by se měl vykonat v momentě objevení neočekávaného průběhu grafů, resp. příliš vysokých hodnot na něm vykreslených.

3.1.2 Návrh a tvorba uživatelského rozhraní

Pro návrh a tvorbu uživatelského rozhraní je určen Interface Builder. V tomto nástroji je možné vytvořit nákres jednotlivých obrazovek či komponent bez toho, aniž by vývojář musel napsat jediný řádek kódu. Jeho hlavní výhodou oproti vytváření vzhledu v kódu aplikace je ta, že vzhled obrazovky je vidět okamžitě, bez nutnosti spuštění aplikace. Jednotlivé obrazovky je možno vytvářet zvlášť v souborech „xib“. Dále je možné vytvoření „storyboardu“, který podobně jako xib, slouží k návrhu obrazovek. Narozdíl od něj jich však může obsahovat více a dále je propojovat. Při práci ve větších týmech se však jeho používání může stát problematické, pokud nejsou jednotlivé případy užití rozumně rozděleny do více storyboardů. Pokud by na jednom souboru mělo pracovat více lidí, je slučování změn takřka nemožné.



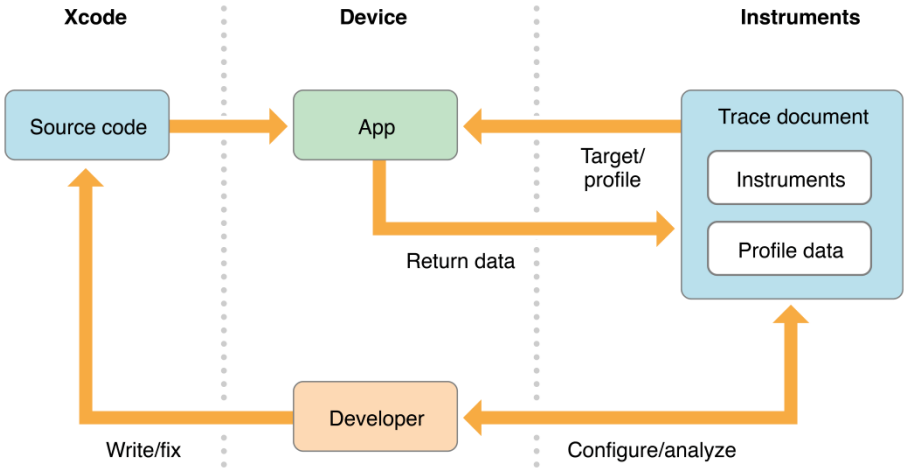
Obrázek 3.1: Interface Builder s otevřeným storyboardem, v kterém lze vytvářet design obrazovek a nastavovat jejich propojení.

Celá tvorba probíhá stylem „drag and drop,“, čili taháním komponent na konkrétní místo, na kterém se pak zvětší či zmenší podle potřeby. Jednotlivé prvky se vybírají z pravé dolní nabídky (obr. 3.1). Přidat lze jakoukoliv komponentu, ať už se jedná o komponentu nativní nebo vytvořenou vývojářem. Pro použití vlastní komponenty je potřeba nejdříve napozicovat základní komponentu `UIView`. U té pak nastavíme konkrétní třídu, která má být pro prvek použita. Pokud chceme mít vlastní komponentu vykreslenou v Interface builderu tak, jak by vypadala v aplikaci, je potřeba ji deklarovat s direktivou `@IBDesignable`. Skrz uživatelské rozhraní lze nastavit i parametry jednotlivých prvků, ať už se jedná o písmo, barvu, velikost nebo barvu pozadí. Pokud nějaké parametry chybí, je možné v kódu aplikace komponentu rozšířit o danou vlastnost, která bude deklarována s direktivou `@IBInspectable` [8].

Ve storyboardech je možné nastavit interakci k zobrazení dalších obrazovek. To se děje pomocí tzv. Segues, které definují jakým způsobem se má nová obrazovka zobrazit. Ke každému lze nastavit identifikátor. Ten slouží k identifikaci při vyvolání Segue. Díky tomu lze učinit další akce, které mají před zobrazením nové obrazovky proběhnout (např. naplnění nové obrazovky daty)[8].

3.1.3 Autolayout

Autolayout dynamicky počítá velikost a pozici všech prvků obrazovky podle omezení, které mají nastaveny. Například v případě, že je tlačítko, které má být vždy na spodní straně obrazovky s pevnou výškou a roztáhnuté na šířku displeje, přičemž obrázek nad ním je vždy 5 obrazových bodů od okraje obrazovky i od tlačítka. Při změně velikosti obrazovky (např. rotace telefonem) se pozice i velikost přepočítají tak, aby vždy platily dané pod-



Obrázek 3.2: Diagram pracovního postupu při používání nástroje Instruments, který slouží k odhalení přílišného využívání prostředků.[1]

mínky omezení. Výsledkem je, že tlačítko je roztažené do šířky podle velikosti obrazovky a jeho výška zůstala stejná, jak bylo definováno. Obrázek nad ním změnil velikost, přičemž odsazení od krajů a tlačítka je chtěných 5 pixelů. Tento postup dovoluje tvořit uživatelské rozhraní, které dokáže reagovat na externí i interní změny.

Externí změny

- Rotace zařízení
- Změna velikosti status baru
- Podpora size classes
- Podpora různých velikostí displejů

Interní změny

- Změna zobrazovaného obsahu
- Aplikace podporuje internacionalizaci
- Aplikace podporuje dynamic types

Nastavení jednotlivých omezení (constraints) probíhá vždy mezi dvěma komponentami kromě případu, kdy se pro prvek nastavuje pevná šířka nebo výška. Lze nastavit zarovnání (např. na střed) v nadřazeném kontejneru, vzdálenost od jiného prvku nebo velikost korespondující s jiným prvkem. Těchto omezení pro každý prvek musí být nastaveno tolik, aby systém mohl vždy dopočítat velikost a pozici zobrazovaného elementu.

[1] Zdroj: *Apple Inc.:* Guides and Sample Code – Instruments User Guide[2]
https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/

3.2 Programovací jazyk

Aplikace pro iOS lze primárně vyvíjet ve dvou jazycích. Těmi jsou Objective-C a Swift. Objective-C je objektově orientovaný jazyk vytvořený na počátku 80. let minulého století. Je nadstavbou jazyka C, která umožňuje zasílání zpráv podobným systémem jako v jazyce Smalltalk [10]. Jedná se o dynamicky typovaný jazyk. Swift je multi-paradigmatický, kompilovaný, open source programovací jazyk od společnosti Apple. Využívá moderních konceptů a syntaxe. Běžně se nepracuje s ukazateli, ikdyž je lze v případě potřeby využít. Oproti Objective-C používá tečkovou notaci k volání metod. Dále byly zavedeny jmenné prostory. Velkým rozdílem je, že se jedná o silně typovaný jazyk. Díky tomu může být přeložený kód značně optimalizován. Dále se tímto například odstraní při volání metody protokolu nutnost zjištění, zda je volaná metoda dostupná. Pokud by tak v Objective-C nebylo učiněno, mohlo by to zapříčinit pád aplikace.

3.3 Nástroje pro správu knihoven

Pokud vývojář chce použít knihovny třetích stran, je vhodné pro jejich správu použít nějaký z dostupných nástrojů. Mezi nejpoužívanější se řadí CocoaPods a Carthage. S příchodem jazyka Swift 3 byl vydán Swift Package Manager, který však v tuto chvíli stále není dostupný pro vývoj na iOS [3]. Proto se tato podkapitola zaměří na první dva zmíněné nástroje.

3.3.1 Carthage

Carthage je decentralizovaný nástroj pro správu závislostí. Jedná se o velmi jednoduchý nástroj. Základní použití se dá popsat v několika krocích:

1. Vytvoření „Cartfile“, ve kterém bude seznam knihoven, které budou použity v projektu.
2. Použití příkazu `carthage update`. Tím proběhne stažení balíčků a předkompilace.
3. Import předkompilovaných balíčků do projektu.

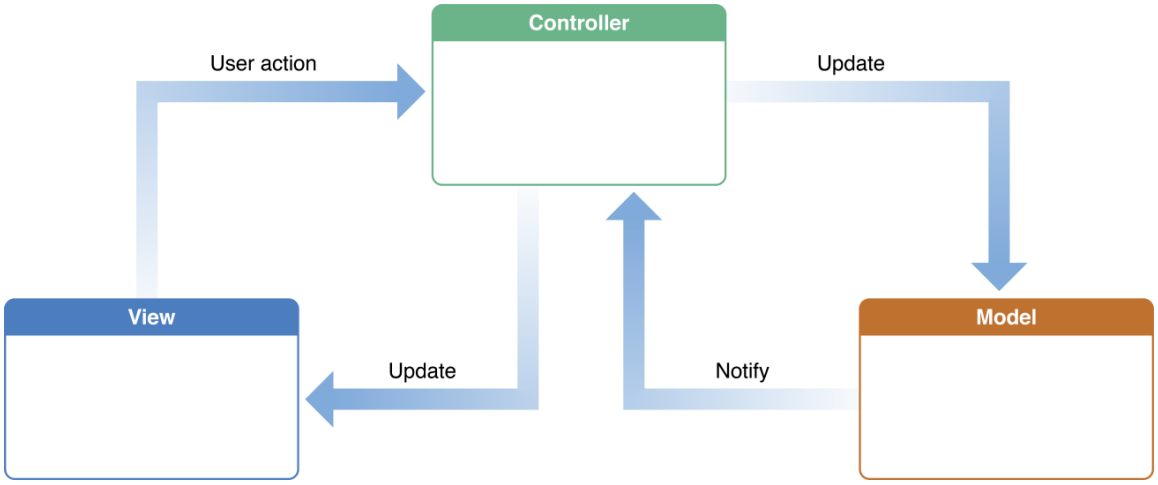
Jeho hlavní výhodou je, že do projektu nezanáší žadné soubory navíc, kromě chtěných knihoven. Nevýhodou je horší vyhledávání.

3.3.2 CococaPods

CocoaPods je pravděpodobně nejpoužívanější nástroj pro správu knihoven. Narozdíl od Carthage je centralizovaným nástrojem. Umožňuje snadné vyhledávání knihoven. Jeho nastavení je při základním používání velmi podobné předchozímu nástroji, a to:

1. Vytvoření Podfile s definovanými knihovnami, které mají být dostupné v jednotlivých cílech sestavení.
2. Použití příkazu `pod install`. Tím se stáhnou balíčky, vytvoří pracovní prostředí a nastaví se závislosti.

Protože 2. krok nastavuje i závislosti, není potřeba importovat balíčky do projektu manuálně, jako tomu bylo v předchozím případě. Dalším rozdílem je, že jsou staženy zdrojové kódy balíčků, které se kompilují až při překladu samotné aplikace.



Obrázek 3.3: Diagram návrhového vzoru Model View Controller[1], který se často používá při tvorbě aplikací na platformě iOS. Je velmi snadno pochopitelný. Vzhledem k jeho benevolenci může velmi snadno vzniknout příliš velký kontrolér.

3.4 Apple Developer Program

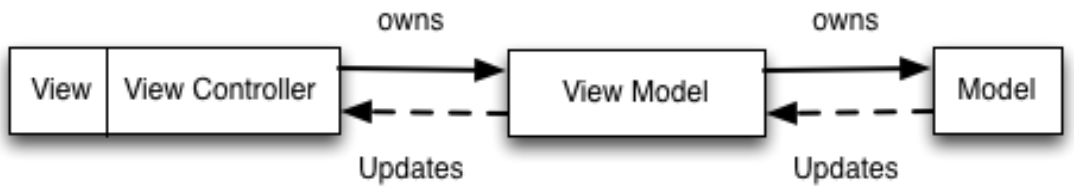
Apple Developer Program je nedílnou součástí vývoje aplikací pro iOS. Avšak do okamžiku, kdy má být aplikace testována v rámci beta testování, není podmínkou být jeho členem. Pakliže si členství vývojář zaplatí za 99 $, dostane možnost vyvíjet aplikace s podporou iCloud, Game Center nebo nakupování uvnitř aplikace [4]. Kromě toho je možné v případě problému s používáním nativních knihoven dvakrát požádat o podporu zdarma. Další požadavky o pomoc jsou již zpoplatněny. Dále umožňuje aplikaci publikovat v aplikaci AppStore pomocí iTunesConnect, což slouží ke správě aplikací. Pokud by měla být aplikace publikována in-house (interně), v rámci společnosti, je nutné si zaplatit program Apple Developer Program Enterprise za 299 $ za rok.

3.5 Architektura mobilních aplikací

Dle Cocoa Core Competencies[1] je při vývoji aplikací pro iOS vhodné použít návrhový vzor známý jako Model View Controller (MVC). Jak je vidět na obrázku 3.3 a zřejmé i z názvu, obsahuje tři typy objektů a těmi jsou kontrolér, view a model. Nositelem dat je model, ať už se jedná o ta, která byla stažena ze vzdáleného serveru, nebo byla uložena lokálně. Model je upravován kontrolérem. Pokud přijdou nová data ze serveru, model upozorní kontrolér o změně. Ten má pak za úkol data zpracovat a nastavit informace k zobrazení na obrazovce (View). Přes obrazovku jsou kontroléru posílány informace o uživatelské interakci. Na základě nich je kontrolérem upraven model. Nevýhodou tohoto vzoru je benevolence, díky které programátor může dát kontroléru příliš mnoho odpovědností, což značně sníží čitelnost kódu a jeho testovatelnost.

Pokud vývojář chce tomuto zamezit, může použít jiný návrhový vzor. Mezi používanými je například Model–View–Viewmodel (MVVM). Jeho diagram je vyobrazen na obrázku 3.4,

[1] Apple Inc.: *Guides and Sample Code – Cocoa Core Competencies* [2] https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html#//apple_ref/doc/uid/TP40008195-CH32-SW1



Obrázek 3.4: Model-View-ViewModel (zdroj: [6]) – Návrhový vzor, který lépe rozděluje odpovědnosti jednotlivých částí, než je tomu u MVC. Tím se zamezí vznik příliš rozsáhlé implementace ViewControlleru.

který pochází z článku „Úvod do MVVM“ [6]. View i ViewController zde označuje prostředek, který zobrazuje informace uživateli, přičemž ViewController vlastní hierarchii View (dále View i ViewController bude zmiňováno jako View). V případě události uživatelské interakce View informuje ViewModel o provedené akci. Pokud by na jejím základě měla nastat nějaká změna v datech, ViewModel upraví Model, který vlastní. Na základě proběhlých změn ViewModel přenastaví View. ViewModel ani Model by neměly pracovat s žádnými prostředky z knihovny UIKit. To pomůže přenositelnosti kódu na jinou platformu a zároveň testovatelnosti.

Kapitola 4

Návrh aplikace

Tato kapitola popisuje návrh aplikace. Obsahuje popis případů užití, návrh databáze pro ukládání jednotlivých záznamů. Je zde i navrženo použití architektury aplikace, která by měla být dodržena během samotné implementace.

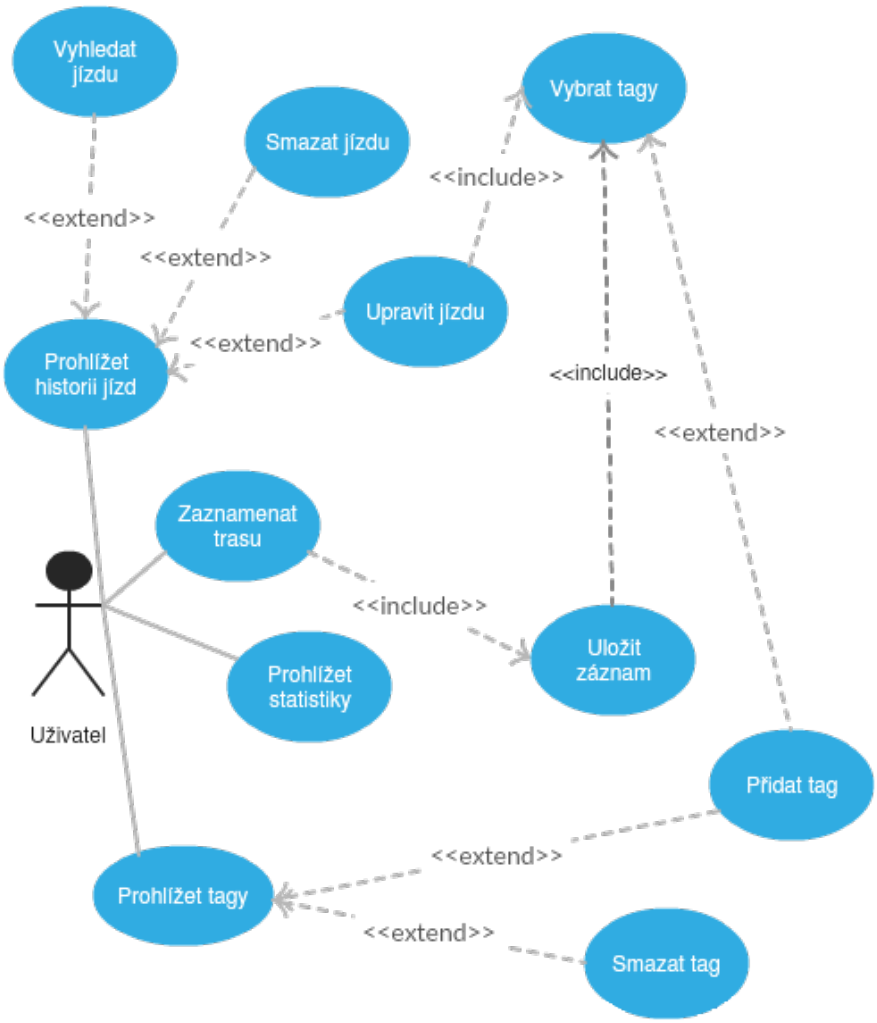
4.1 Případy užití

Na obrázku 4.1 se nachází diagram případů užití aplikace. Mezi hlavní funkcionalitu patří zaznamenávání trasy, bez níž by aplikace ztrácela svůj smysl. Ta obsahuje mechanismus pro detekci automatické pauzy, který lze podle uvážení uživatele vypnout. Zaznamenaná trasa se průběžně vykresluje na mapu. Na té je zobrazena i aktuální poloha uživatele, na kterou se mapa vycentruje při každé změně polohy kromě případu, kdy byla provedena interakce s mapou. Po interakci není centrování prováděno po dobu následujících 10 sekund.

Zaznamenanou jízdu lze uložit, přičemž je uživatel vyzván k vyplnění jména, popisu a tagů. Jméno je předvyplněno řetězcem vytvořeným ze dne, ve kterém zaznamenávání začalo. Vyplnění údajů není povinné. Na této obrazovce bude vykreslena trasa, kterou řidič ujel. Při výběru tagů je možné vytvářet nové.

Uživatel si dále může prohlédnout historii jízd, v níž je možné vyhledávat podle tagů, jména a popisu. Při psaní do vyhledávacího pole je zobrazena nápověda s tagy, které mají prefix totožný s právě psaným slovem. Pokud žádný takový tag neexistuje, pak se nápověda nezobrazí. Při kliknutí na tag se zbývající část názvu doplní. Při úpravě textu ve vyhledávacím poli se průběžně zobrazují výsledky hledání. Z historie je možné záznam smazat, pomocí `swipe` gesta a kliknutí na potvrzovací tlačítko. Z historie i z výsledků hledání je možné si zobrazit detail záznamu. V něm je obsažena mapa s vykreslenou trasou. Její barva se mění podle rychlosti, jakou byl daný úsek projet. Dále obsahuje informace, které uživatel při ukládání vyplnil a graf času, který byl strávený v rychlostech. Z detailu je možné si jízdu upravit. K tomu je použita totožná obrazovka, jaká slouží k ukládání záznamu.

V aplikaci je možné zobrazit statistiky uživatele. Mezi ně patří celková délka zaznamenaných tras, celkový čas strávený zaznamenáním. V případě používaní automatické pauzy jsou z ní vyloučeny přestávky, které byly detekovány. Nachází se zde i graf nájezdu kilometrů. Je možné si v něm zobrazit data za poslední měsíc, týden nebo rok. Při volbě daného úseku se zároveň uživatel dozví o průměrném denním nájezdu, který je spočítán pouze ze dnů, ve kterých byl vytvořen nějaký záznam.

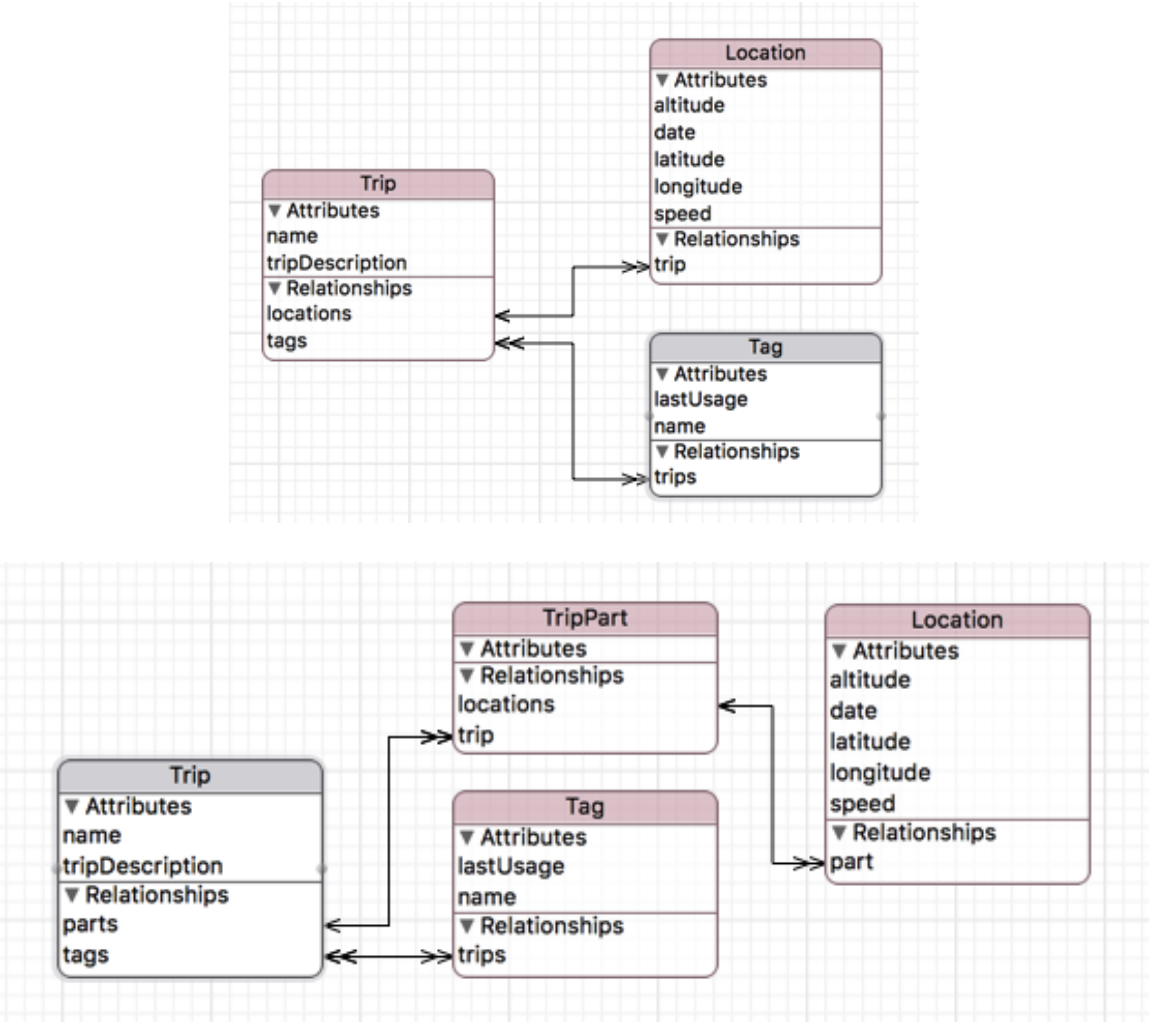


Obrázek 4.1: Diagram případů užití aplikace

4.2 Databáze

Databáze se v průběhu iterací vývoje několikrát upravovala. První verze (obr. 4.2 nahoře) sloužila pouze k uložení zaznamenané trasy. Její schéma obsahuje pouze tři tabulky. Jednou z nich je „trip“, který označuje jeden záznam trasy. Jeho atributy jsou jméno a popis. Tato tabulka může vlastnit neomezené množství nasbíraných lokací (tabulka Location). Ta reprezentuje entitu pozice, na které se řidič vyskytoval. Obsahuje přesný čas zíkání pozice v atributu „date“. Dále jsou zde atributy označující zeměpisnou šířku a výšku. Třetí tabulka nese název „Tag“. Ta uchovává štítky, které lze přidat k záznamům jízd. Každý štítek obsahuje informaci o jméně a času posledního výběru. Toto schéma je plně dostačující, pokud by aplikace neměla možnost pozastavování záznamu.

V druhé verzi (obr. 4.2 dole) byl upraven způsob ukládání lokací. Přibyla tabulka „TripPart“, která reprezentuje úsek záznamu. Ten je vždy vytvořen při spuštění či obnovení zaznamenávání během aktivní přestávky. Tato tabulka je v relaci s „Trip“, která již není přímo vázaná na „Location“. Namísto toho byl vytvořen vztah mezi „TripPart“ a „Location“ s kardinalitou 1:N.



Obrázek 4.2: Diagram entit. **nahoře:** verze bez podpory přerušování zaznamenávání; **dole:** verze s podporou přerušovaní, (jednoduchá šipka značí kardinalitu 1, zdvojená kardinalitu N).

4.3 Architektura aplikace

Implementace aplikace by měla být vytvářena podle jistých pravidel, která napomáhají k tvorbě kvalitního kódu. K tomu může být použit některý ze známých návrhových vzorů, které byly zmíněny v kapitole 3.5. Vzhledem k aktuálním trendům jsem zvolil MVVM. S kolegy, se kterými jsem řešil komerční projekty, jsme se tuto architekturu, na základě debaty a studia článků dalších vývojářů, rozhodli rozšířit o koordinátor. Ten by měl řešit navigaci. [9]. V případě iOS se tím odstraní z třídy `UIViewController` odpovědnost za prezentování dalších obrazovek.

Koordinátor je v aplikaci držen instancí třídy dědící z `UIViewController`, jehož instance je získána pomocí Storyboard Segue, nebo instanciací ze Storyboardu. Ta je následně naplněna požadovaným ViewModelem. Výhodou tohoto řešení je, že zavřením obrazovky dojde k dealokaci instance koordinátoru.

K dodržování této architektury pomáhají části knihovny FuntastyKit, zejména se jedná o protokoly různých druhů koordinátorů:

Obrázek 4.3: Použitá paleta barev zleva: klikatelné prvky s interakcí, neklikatelné prvky (pozadí tagů), běžný text, méně důležité informace

- `PushCoordinator` – Zajistí, aby obrazovka byla zobrazena pomocí `UINavigationControlleru`, který vloží UIViewController do svého zásobníku obrazovek.
- `ModalCoordinator` – Obrazovka se prezentuje modálně, čili přes celou obrazovku, bez možnosti řízení navigace.
- `PushModalCoordinator` – ViewController je zobrazena modálně, avšak s vlastním `UINavigationController`. Díky tomu je možné využívat jeho výhody pro další průchod obrazovkami (zobrazení detailu apod.).

4.4 Návrh uživatelského rozhraní

Aplikace je tvořena převážně pomocí nativních prvků. Jedinou výjimkou je nápověda tagů při vyhledávání v historii jízd. Grafický návrh aplikace je dostupný ve složce design na přiloženém médiu. Na obrázku 4.3 se nachází barevná paleta, jejíž barvy byly použity u jednotlivých prvků uživatelského rozhraní. Tmavě modrá je použita pro prvky podporující uživatelskou interakci. Světlejší varianta barvy je použita pro tagy, na které nelze kliknout. Barva textu je černá, přičemž méně důležité informace jsou šedé.

Kapitola 5

Implementace

Kapitola se zabývá implementací jednotlivých částí aplikace v programovacím jazyku Swift 3. Dále jsou v ní popsány problémy, které se během ní objevily a jejich řešení. Pořadí kapitol koresponduje se zvoleným postupem tvorby jednotlivých obrazovek.

5.1 Tvorba uživatelského rozhraní

Stejně jako pro návrh aplikace byl pro tvorbu uživatelského rozhraní použit InterfaceBuilder. V něm bylo vytvořeno několik storyboardů. V hlavním se nachází kořenový ViewControllerem typu `UITabBarController`, díky kterému se zobrazí takzvaný „TabBar“, čili lišta se záložkami. Ty jsou v aplikaci tři, a to:

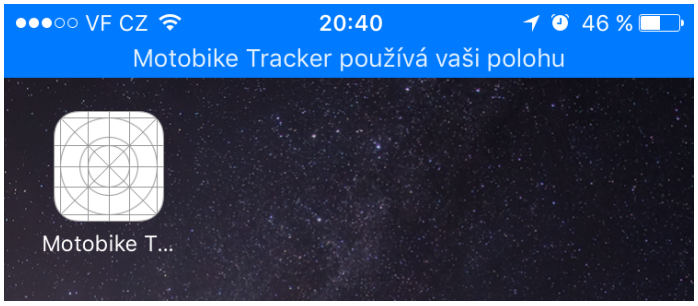
- Zaznamenávání (záložka Tracking)
- Historie záznamů (záložka Logbook)
- Profil jezdce (záložka Profil)

Každá z těchto záložek (obrazovek) je vytvořena ve vlastním storyboardu. Obrazovka má obsahovat navigační lištu. Ta je při použití základního nastavení na obrazovce zobrazena, je-li daný `UIViewController` prezentován pomocí `UINavigationController`. Toto navigační rozhraní umožňuje efektivně prezentovat data a usnadňuje uživateli navigaci v tomto obsahu [1]. Na každý z těchto ViewControllerů typu `UINavigationController` je v hlavním storyboardu vytvořena reference, která je přiřazena do `UITabBarController`.

5.2 Získání lokace uživatele

Tato část se zabývá potřebnou konfigurací pro přístup k polohovým službám. Dále je popsáno vytvoření prostředku, který zapouzdřuje nativní knihovnu, která slouží k doručování událostí týkajících se lokace.

[1] Zdroj: *Apple Inc.:* API Reference – UINavigationController [1] - https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/



Obrázek 5.1: Při běhu aplikace na pozadí se spuštěným zaznamenáváním je uživatel informován o používání polohy změnou barvy a velikosti stavového řádku, kterému přibyl text s touto informací

5.2.1 Konfigurace

Aby aplikace mohla používat polohové služby, je nutné požádat uživatele o povolení k jejímu přístupu. Pro zobrazení takovéhoto požadavku je potřeba upravit soubor `Info.plist` (Information Property List, čili seznam vlastností aplikace). Jedná se o soubor, který obsahuje XML strukturu popisující různé aspekty svazku (Bundle) aplikace[2]. Do kořenového uzlu tohoto souboru, byl přidán klíč `NSLocationWhenInUseUsageDescription`, v jehož hodnotě jsou důvody popisující, proč je vyžadován přístup k aktuální poloze během užívání aplikace. Tento text se pak zobrazí ve zmíněném požadavku. Dále je nutné přidat klíč `UIRequiredDeviceCapabilities`. Jeho hodnotou je pole řetězců označujících funkce, bez které aplikace nemůže fungovat. V případě této aplikace se jedná o [8]:

- `location-services` – přidává se, pokud aplikace obecně vyžaduje polohové služby,
- `gps` – přidává se, pokud je potřeba mít přesnou polohu, což je případ této aplikace.

Aplikace má zaznamenávat polohu, ikdyž nebude na popředí. K tomu je potřeba do seznamu vlastností přidat klíč `UIBackgroundModes` s hodnotou obsahující řetězec `location` [8].

Při běhu na pozadí se spuštěným záznamem se zobrazí modrý stavový řádek s informací, že aplikace používá polohu (viz obrázek 5.1).

5.2.2 Vytvoření služby pro průběžné získávání polohy

K získání polohy uživatele slouží v iOS knihovna CoreLocation. Ta poskytuje službu nejen pro určení geografické lokace, ale i nadmořské výšky, orientace nebo polohy zařízení vzhledem k blízkému iBeacon. Knihovna používá veškeré dostupné hardwarové vybavení, včetně Wi-Fi, GPS, Bluetooth, magnetometru, barometru a celulárního hardwaru pro shromažďování dat[3].

Pro snadné znovupoužití byl nejprve vytvořen protokol `LocationManager`, ten je implementován třídou `LocationManagerImpl`. Čtyři metody tohoto protokolu slouží následovně:

[2]Zdroj: Apple Inc.: *Guides and Sample Code – Information Property List Key Reference* [2] – https://developer.apple.com/library/content/documentation/General/Reference/InfoPlistKeyReference/Articles/AboutInformationPropertyListFiles.html

[3]Zdroj: Apple Inc.: *API Reference – CoreLocation* [1] – https://developer.apple.com/reference/corelocation/

- `func request(authorization: LocationManagerAuthorization)`

  Slouží pro povolení přístupu k poloze uživatele. Tato funkce zapouzdřuje metody třídy `CLLocationManager` `requestAlwaysAuthorization` a `requestWhenInUseAuthorization`.

- `func canTrackLocation() -> Bool`

  Pomocí třídních metod `CLLocationManager` detekuje, zda-li zaznamenávání polohy je v nastavení telefonu povolené a zároveň je schválen přístup k polohovým údajům.

- `func startObservingLocation(withHandler handler: @escaping (CLLocation) -> (Void), byObserver observer: AnyObject`

  Touto metodou se spustí sledování změn polohy. Při každé změně jsou zavolány všechny „Closures“, které byly touto metodou zaregistrovány k získání aktuální lokace a nebyly odregistrovány následující metodou.

- `func stopObserving(byObserver observer: AnyObject)`

  Tato metoda odstraňuje rutinu, která byla zaregistrovaná objektem předaným v parametru.

Instanci třídy `CLLocationManager` je možné nastavit požadavek na přesnost. S vysokou požadovanou přesností roste spotřeba baterie[4]. Z tohoto důvodu je nutné nastavit přesnost optimálně vzhledem k požadavkům aplikace. V případě rychlé jízdy na rovinkách při používání mé aplikace je vyžadována nižší přesnost, než je tomu při pomalejší jízdě ve městech, ve kterých se častěji vyskytují semafory a kolony nebo místech, ve kterých je větší počet zatáček. Proto je instanci třídy `CLLocationManager` upravován atribut `desiredAccuracy`, kterým se určuje požadovaná přesnost. Ta je nastavena při změně lokace, a to na hodnotu rychlosti v metrech za sekundu, pokud je rychlost větší než 5 m/s.

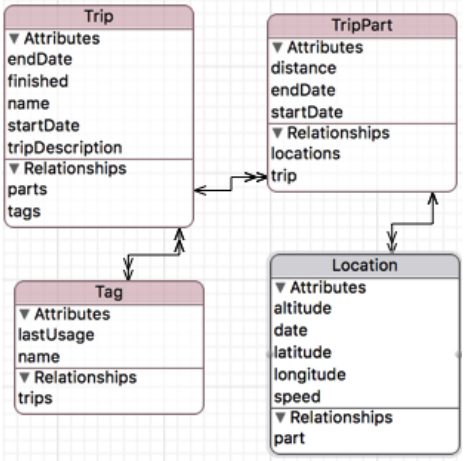
Instance třídy `CLLocationManager` obsahuje atribut `distanceFilter`, který určuje minimální rozdíl vzdálenosti pro získání nové polohy. To znamená, že instance nebude informovat o změně lokace do momentu, než vzdálenost nové polohy od poslední získané bude větší než nastavená minimální hodnota rozdílu. Pomocí této vzdálenosti lze upravit frekvenci aktualizací. Nastavení nemá vliv na spotřebu baterie, ale může pomoci redukovat počet uložených změn lokací v průběhu zaznamenávání. Hodnota minimálního rozdílu vzdáleností je upravena při aktualizaci polohy, a to tak, že pokud je rychlost menší než 30 km/h, pak je minimální vzdálenost nastavena na 5 metrů. Při vyšší rychlosti je hodnota nastavena na 30 metrů. Nižší hodnota je atributu `distanceFilter` přiřazena i při změně směru jízdy o více než 15°, jehož změnu je možné pomocí `CLLocationManager` možno sledovat.

5.3 Ukládání dat

K perzistentnímu ukládání dat je možno na iOS použít několik způsobů [7]:

- UserDefaults – nabízí způsob, jak snadno uložit malé množství dat. Pro větší objemy jsou vhodné jiné technologie. Hodí se například pro ukládání nastavení aplikace.
- Keychain – slouží k ukládání choulostivých informací (např. hesla, certifikáty)

[4] Apple Inc.: *API Reference – CoreLocation* [1] https://developer.apple.com/reference/corelocation/



Obrázek 5.2: Datový model vytvořený pro knihovnu CoreData

- Databáze SQLite
- CoreData – knihovna, která slouží pro správu objektových objektů modelu v aplikaci. Poskytuje obecná a automatizovaná řešení běžných úloh spojených s životním cyklem objektu a řízením grafů objektů, včetně persistence [5].
- Soubor v souborovém systému

Vzhledem k tomu, že aplikace bude obsahovat velké množství dat, jeví se jako nejvhodnější způsob ukládání záznamu jízd knihovna „CoreData“ nebo databáze „SQLite“.

Protože je vytváření a úprava modelu snadnější s pomocí knihovny `CoreData`, rozhodl jsem se pro ní. Její používání může ulehčit knihovna AERecord, která zapouzdřuje některé funkce knihovny CoreData, a tím dělá její používání snadnější.

5.3.1 Datový model

V průběhu implementace byly některé tabulky navrženého databázového modelu rozšířeny o atributy, které sice způsobily redundanci dat, ale významně zrychlily zobrazování grafů, ve kterých se s těmito daty pracuje. Výsledný datový model je zobrazen na obrázku 5.2

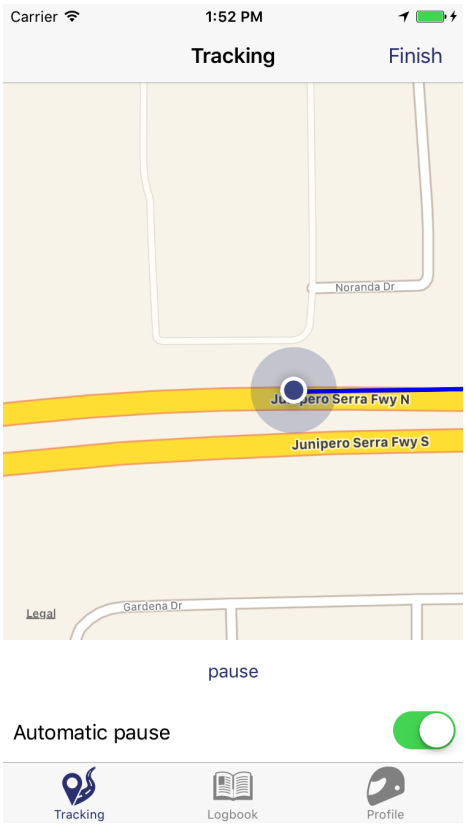
5.4 Zaznamenávání jízdy

Obrazovka je pro celou aplikaci nejdůležitější, neboť jejím úkolem je spuštění zaznamenávání jízdy. Její jednoduchý vzhled je rozdělen do dvou částí, jak je vidět na obrázku 5.3. V horní části lze sledovat aktuální polohu na mapě, na které se zároveň vykresluje ujetá trasa. Spodní část slouží k ovládání záznamu.

5.4.1 Ovládací prvky, získání polohy a automatická pauza

Spodní část obrazovky slouží k ovládání a nastavení zaznamenávání. Je vytvořena pomocí `UIStackView`, které obsahuje dva řádky. První z nich obsahuje tlačítko pro ovládání zazna-

[5] Apple Inc.: *Guides and Sample Code – Core Data Programming Guide* [2]
https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData/index.html#//apple_ref/doc/uid/TP40001075-CH2-SW1



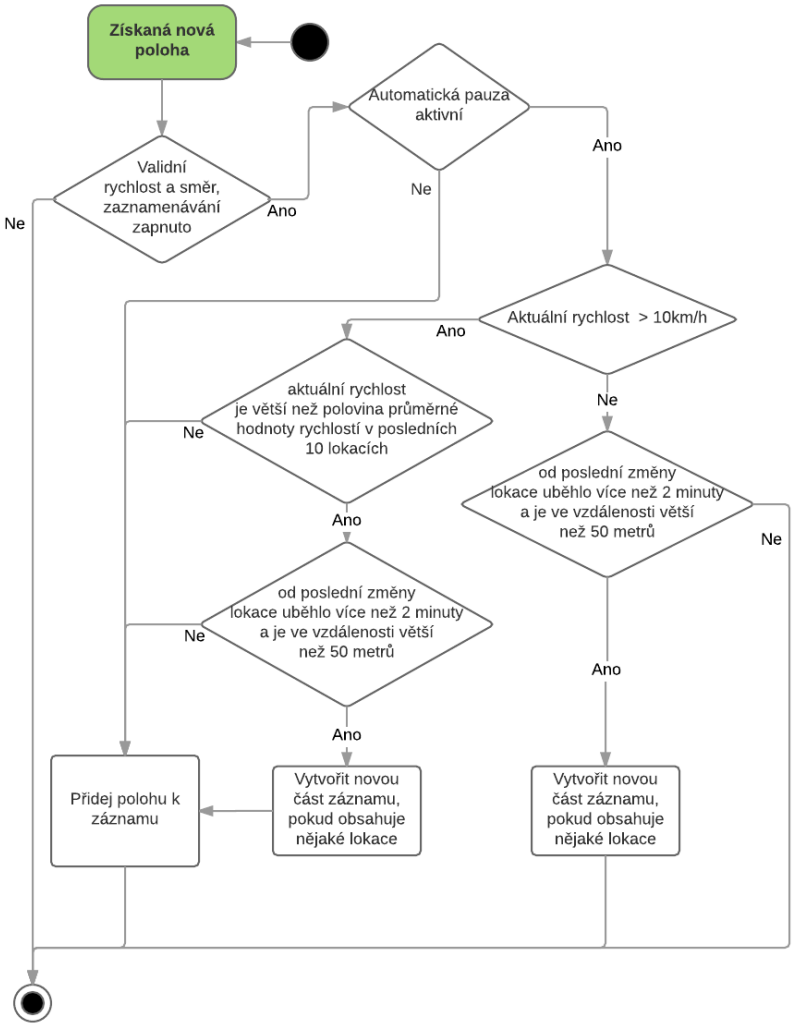
Obrázek 5.3: Obrazovka zaznamenávání trasy při zapnutém zaznamenávání.

menávání. Druhý řádek tvoří `UILabel` s popisem funkce a přepínač pro vypnutí či zapnutí automatické pauzy.

`UIStackView` byla zvolena z důvodu, že se jedná o novou komponentu, dostupnou od iOS 9.0, která vyhovuje potřebám na obrazovce zaznamenávání jízdy. Poskytuje rozhraní pro sestavení kolekce ve sloupci nebo v řádku. V případě přidání možnosti používat aplikaci při orientaci na šířku (režim landscape) by bylo nutné přeuspořádávání prvků při každé změně orientace. To by bylo s pomocí této komponenty, resp. při použití několika zanořených, velmi snadné. Její rozhraní je oproti `UIView` rozšířeno o nastavení osy, která určuje, zda se prvky mají umisťovat pod sebe nebo za sebe. Dále je rozšířeno o typ jejich rozložení, zarovnání a vzdálenosti mezi nimi.

Tlačítku v prvním řádku `UIStackView` se podle stavu zaznamenávání mění titulek a funkcionalita. To zařizuje ViewModel dané obrazovky. Po spuštění aplikace tlačítko nese název „Start“. Po kliknutí na něj se spustí procedura ViewModelu, která inicializuje vše potřebné k započetí zaznamenávání, v čemž spočívá vytváření nových objektů `Trip` a TripPart. Dále zde probíhá registrace sledování událostí změny polohy. Zároveň se změní text tlačítka na „Pause“. Jak název napovídá, jeho funkcí je nyní pozastavení záznamu. ViewModel se při této akci odhlásí z odběru událostí změny polohy, aby nebyla zbytečně zatěžována baterie telefonu a tlačítku se změní název na „Continue“. Ten označuje akci pro pokračování v záznamu. Při ní se oproti prvotnímu spuštění záznamu inicializuje pouze nový objekt `TripPart`, který bude v relaci s aktuální instancí třídy `Trip`.

Druhý řádek obsahuje popisek „Automatic Pause“ a nativní přepínač `UISwitch`, sloužící k vypínání a zapínání automatické pauzy. Jeho aktivace mění proces ukládání nově obdržené



Obrázek 5.4: Zaznamenávání trasy – vývojový diagram popisující události konané při aktualizaci polohy, která obsahuje i údaje o aktuální rychlosti a směru pohybu.

lokace, jak je zřejmé na vývojovém diagramu (obr. 5.4). Při každém získání nové lokace se kontroluje, zda-li struktura `CLLocation` obsahuje validní hodnoty pro směr a rychlost pohybu.

Součástí navigační lišty je tlačítko „Finish“ sloužící pro ukončení zaznamenávání. Při stistku je poslána zpráva koordinátoru, který zajistí zobrazení obrazovky pro uložení jízdy. Ta je zabalena v `UINavigationController`, který je pak zobrazen modálně (obrazovka se vysune od spodního okraje displeje).

5.4.2 Mapa a vykreslování trasy

Pro zobrazení mapy bylo použito `MKMapView` z nativní knihovny MapKit. K jejich používání je nutné aktivovat položku Maps v Capabilities, kterou lze najít v nastavení projektu. Okno s mapou zobrazuje lokaci uživatele, pokud je dostupná. To je umožněno nastavením atributu `showsUserLocation` na kladnou hodnotu.

ViewController této obrazovky implementuje protokol `MKMapViewDelegate`, díky kterému je možné ho nastavit jako delegát mapView. Z tohoto rozhraní jsou implementovány dvě metody:

- `func mapView(_ mapView: MKMapView, didUpdate userLocation: MKUserLocation)`
  Tato metoda je zavolána vždy, pokud se změní pozice uživatele. Implementace centruje mapu na anotaci označující polohu uživatele, pokud však s ní v posledních 10 vteřinách uživatel nepohnul.

- `func mapView(_ mapView: MKMapView, rendererFor overlay: MKOverlay) -> MKOverlayRenderer`
  Slouží k inicializaci a nastavení `MKOverlayRenderer`, který má na starost vykreslování obsahu overlays. Je volána vždy, když se přidá nový overlay. Pro vykreslování trasy během zaznamenávání je jako renderer použit `MKPolylineRenderer`. Tomu je nastavena šířka a barva čáry, kterou bude vykreslovat.

Vykreslení trasy proběhne vždy, když nová lokace uživatele byla přidána k záznamu; kdy se tak děje, je popsáno v kapitole 5.4.1. Pro každou část záznamu je vytvořen `MKPolyline` s polohami, které mu jsou přiřazeny.

Posunutí mapou uživatelem je implementováno pomocí třídy `UIPanGestureRecognizer`. Ta slouží k detekci táhnutí prstem. Vytvořený objekt je pak přiřazen do `mapView`, které však již `UIGestureRecognizer` používá k posunu mapou. Kvůli tomu je k instanci přiřazen delegát na viewController aktuální obrazovky, který implementuje metodu protokolu `UIGestureRecognizerDelegate`:

- `func gestureRecognizer(_ gestureRecognizer: UIGestureRecognizer, shouldRecognizeSimultaneouslyWith otherGestureRecognizer: UIGestureRecognizer) -> Bool`

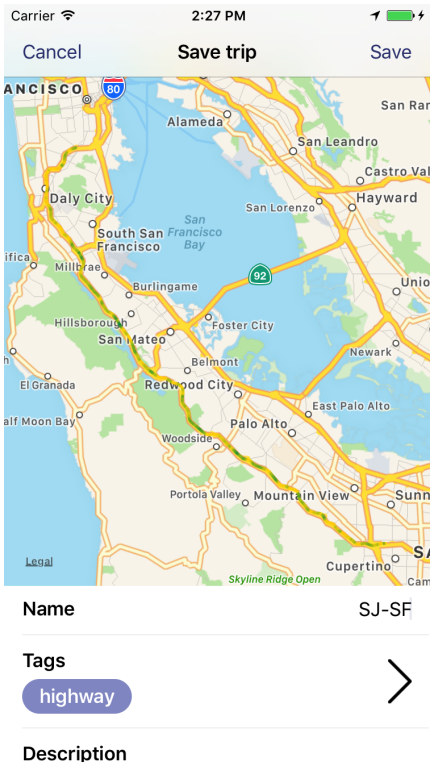
Pokud tato metoda není implementovaná, použije se výchozí chování tohoto protokolu a vrací `false`. Více gest v jednu chvíli pak není povoleno. V tomto případě však je žádoucí opak, neboť by nebylo vytvořené gesto nikdy detekováno. Při vyvolání gesta se pak nastaví příznak, který zamezí centrování mapy. Bez něj by nebylo možné posunovat mapu do jiné části než je ta, ve které se nachází anotace označující aktuální polohu.

5.5 Uložení a úprava jízdy

Pro uložení a úpravu jízdy slouží totožná obrazovka. Základ obrazovky je vytvořen podobně jako v předchozím případu užití. Rovněž je rozdělena do dvou částí – mapové a informační.

V informační části se nastavují údaje o jízdě. Patří mezi ně štítky, nebo-li tagy, jméno a popis. Pole pro jméno je vytvořeno jako `UITextField`, který slouží pro vyplňování jednořádkových vstupů od uživatele. Má předvyplněnou hodnotu, kterou je datum a čas spuštění záznamu, což ulehčí ukládání ve spěchu. Pro popis je zde použita komponenta `UITextView`. Ta umožňuje zobrazit text na více řádcích a zároveň je upravitelná. Její implementace je však složitější, neboť nepodporuje `placeholder`, čili text, který je zobrazen, pokud pole nemá žádnou hodnotu. Toto bylo vyřešeno vložením pevného textu pod dané `UITextView`. Placeholder se pak zobrazuje podle toho, zda-li je popis jízdy prázdný či nikoliv.

Část, která vybízí ke zvolení tagů, je zvláštní v tom, že přes ní celou je roztáhnuté tlačítko `UIButton`. Po kliknutí na něj se zobrazí tabulka s nabídkou výběru tagů. Po výběru se



Obrázek 5.5: Obrazovka pro ukládání a editaci záznamu obsahuje mapu s vyznačenou trasou, pole pro pojmenování záznamu a jeho popis. K záznamu je možné přidat tagy, které se vybírají z nabídky zobrazené po kliknutí na šipku v části „Tags“

zobrazí v buňkách kontejneru `UICollectionView`, který se nachází pod zmíněným tlačítkem. Jsou zarovnány vlevo díky použití knihovny „UICollectionViewLeftAlignedLayout“, která byla získana z CocoaPods. Jednotlivé buňky jsou neklikatelné a obsahují `UIView` s barvou pozadí. Vrstvě (`layer`), která slouží k vykreslování, je pak nastavena vlastnost `cornarRadius` na poloviční výšku buňky, kvůli zaoblení pravého a levého okraje buňky. Zároveň je mu nastaven atribut `masksToBoBounds` na hodnotu `true`. Pokud by nebyl nastaven, buňka by stále vypadala jako obdelník.

Na mapě zobrazené v horní části obrazovky je vykreslena zaznamenaná trasa. Její části se barevně odlišují podle rychlosti, kterou uživatel místo projel. Její implementace je popsána v kapitole 5.7.2.

5.6 Historie jízd

Obrazovka pro zobrazení historie jízd je vyobrazena na obrázku 5.7 vlevo (str. 30). Její hlavní komponentou je `UITableView`, ve které se zobrazují všechny záznamy. Jejich buňky obsahují základní údaje o ujetých jízdách. Layout je vytvořen v souboru xib. Je na nich vyobrazeno jméno, datum, ujetá vzdálenost a tagy. Tagy se zobrazují stejně jako na obrazovce pro uložení záznamu (kapitola 5.5). Jednotlivé elementy mají definované vzdálenosti pomocí omezení z autolayout, a to mezi svými sousedy a kontejnerem, ve kterém jsou vloženy. Díky tomu lze v `UITableView` použít automatický výpočet velikosti buňky, který se nastavuje přiřazením `UITableViewAutomaticDimension` do atributu `rowHeight`. Na obra-

zovce je dostupné vyhledávání, které usnadní dohledávání jízd. Způsob implementace bude popsán dále. Z této obrazovky je možná navigace do detailu některé ze zobrazených jízd. Obrazovka se zobrazí po kliknutí na některou z buněk. Děje se tak pomocí koordinátoru, nad kterým je invokována metoda s parametrem nesoucím viewModel dané buňky.

5.6.1 FetchResultsController

`NSFetchedResultsController` je třída pomocí níž lze sledovat změny v CoreData, proto je vhodná pro použití při získávání dat. Obsahuje generický parametr, který určuje, s jakým typem objektů bude třída pracovat. V případě historie jízd je to `Trip`. Tento kontroler je neinicializován ve ViewModelu, který implementuje protokol `NSFetchedResultsControllerDelegate`. Ten po přiřazení do atributu `delegate` odposlouchává změny skrz daný protokol. Jeho implementovanými metodami jsou:

- `func controller(`
  `_ controller: NSFetchedResultsController <NSFetchRequestResult>,`
  `didChange anObject: Any, at indexPath: IndexPath?, for type:`
  `NSFetchedResultsChangeType, newIndexPath: IndexPath?)`
- `func controllerWillChangeContent(_ controller:`
  `NSFetchedResultsController<NSFetchRequestResult>)`
- `func controllerDidChangeContent(_ controller:`
  `NSFetchedResultsController<NSFetchRequestResult>)`

První metoda je volána vždy, když nastane změna na některém objektu, je smazán nebo vytvořen nový. Je zároveň i volána v případě, že se objekt ve výsledcích z nějakého důvodu posune. Její implementace pak volá delegát ViewModelu s metodou, která je nadeklarována podobně. Místo samotného objektu záznamu však posílá `LogbookCellViewModel`, který ho zabaluje. Druhá je volána tehdy, když má nastat nějaká změna v obsahu a třetí, když změny již byly provedeny. Protokol delegátu ViewModelu rozšiřuje LogbookViewController starající se o tuto obrazovku. Jeho metody jsou:

- `func viewModelWillChangeCells(_ viewModel: LogbookViewModel)`
  V implementaci se invokuje metoda `beginUpdates` tableView ve ViewControlleru, čímž oznamuje, že budou probíhat změny v DataSource
- `func viewModelDidChangeCells(_ viewModel: LogbookViewModel)`
  Při vykonávání této metody je poslána zpráva `endUpdates` objektu `tableView`, čímž oznamuje dokončení aktualizace.
- `func viewModel(_ viewModel: LogbookViewModel, didChange cellViewModel:`
  `LogbookCellViewModel, at indexPath: IndexPath?, for type:`
  `NSFetchedResultsChangeType, newIndexPath: IndexPath?)`
  Pokud je typem změny `insert`, pak je do tabulky vložen objekt na pozici v `newIndexPath`. Pro typ `delete` je smazána buňka `indexPath`. Při změně pozice je metoda volána s typem `move`. Typ `update` přichází v případě potřeby aktualizovat buňky. Pro každý typ je pak volána odpovídající funkce nad instancí třídy `UITableView`.

5.6.2 Mazání jízd

K mazaní buněk ze seznamu jízd je potřeba buňku posunout doleva, čímž se zobrazí tlačítko „Delete“. Po kliknutí na něj se jízda smaže. K vytvoření této funkcionality je potřeba implementace dvou metod protokolu `UITableViewDelegate`, a těmi jsou:

- `func tableView(_ tableView: UITableView, editingStyleForRowAt indexPath: IndexPath) -> UITableViewCellEditingStyle`)
- `func tableView(_ tableView: UITableView, commit editingStyle: UITableViewCellEditingStyle, forRowAt indexPath: IndexPath)`

První metoda vrací hodnotu `delete`. To způsobí, že při gestu posunutí buňky vlevo se zobrazí tlačítko na červeném podkladu s názvem „Delete“. Po kliknutí na něj se zavolá druhá metoda. Ta by měla obsahovat volání funkce pro smazání objektu na daném řádku v tabulce. V tomto případě se jedná o volání metody nad instancí ViewModelu, která smazání záznamu zajistí.

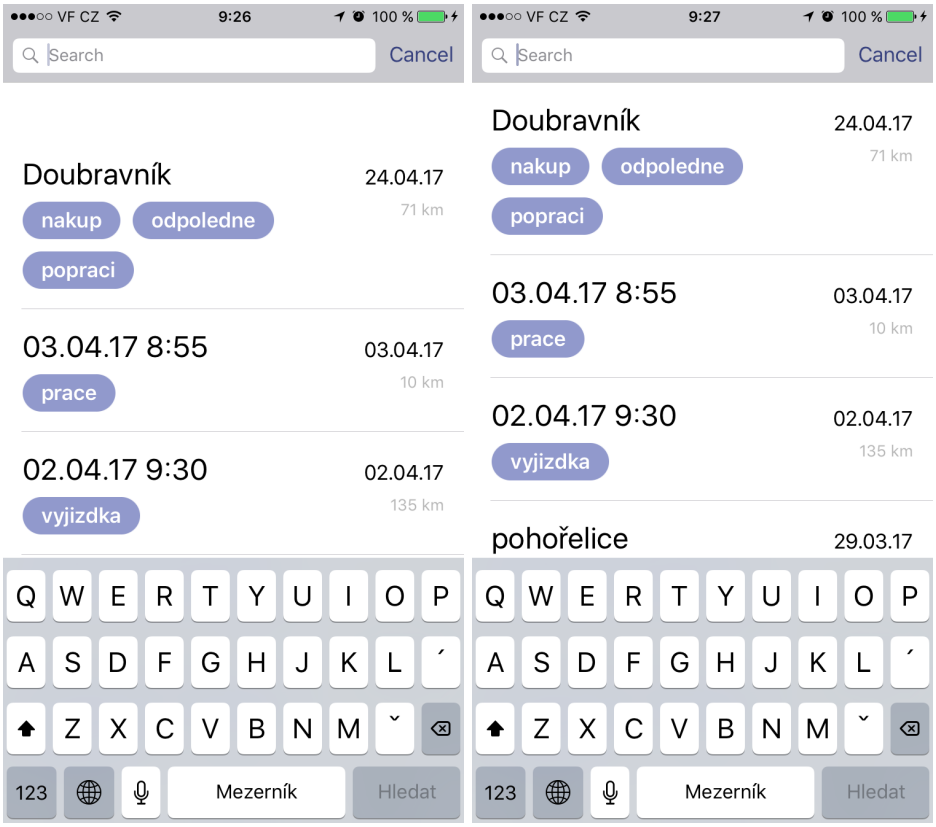
5.6.3 Vyhledávání jízdy

K prezentaci vyhlevacího pole a výsledků byly použity nativní prvky. Mezi ně patří `UISearchBar`, který lze vložit přímo do tabulky. Ten lze získat instanciací `UISearchController`. Pro jeho vytvoření je potřeba `UIViewController`, ve kterém se budou prezentovat výsledky hledání. Proto je vytvořen `LogbookResultsViewController`. Ten je nainstanciován ze Storyboardu, ve kterém byl vytvořen design uživatelského rozhraní. Jeho součástí je i našeptávání tagů, podle kterých je možné vyhledávat.

To se udělá přiřazením vyhledávacího pole do atributu `tableHeaderView`. Díky tomu se pole zobrazí v horní části tabulky, přičemž je spolu s obsahem skrolovatelný. Po kliknutí do vyhledávacího pole se `UISearchBar` přemístí z tabulky na pozici navigačního panelu. Protože view controller historie jízd je prezentován z `UITabBarControlleru`, bylo nutné mu nastavit příznak `definesPresentationContext` na kladnou hodnotu, jinak by se výsledky zobrazovaly s posunutím od vyhledávacího pole (obr. 5.6). Jakmile uživatel začne psát do textového pole `UISearchBar`, je každá aktualizace z ViewControlleru poslána do ViewModelu. Ten začne rozhodovat, jaké výsledky se mají zobrazit na základě hledání. Jednotlivá slova jsou vždy vyhledána v názvech jízd a v popisu.

Dále je rozeznáváno datum pomocí třídy `NSDataDetector` z knihovny Foundation. Díky tomu lze vyhledat jízdu z konkrétního dne. Detektor z předloženého řetězce dokáže rozeznat datum podle nastavení lokalizace operačního systému. Pokud je mu předloženo datum v jiném než očekávaném formátu, pak není detekováno a následně se podle něj vyhledává.

Ve vyhledávacím řetězci, který příjme ViewModel, je zjišťován obsah tagů. Pokud jsou nalezeny, vyfiltrují se podle nich záznamy, které se mají zobrazit. Jízda je z výsledku vyloučena, pokud neobsahuje všechny z detekovaných tagů. Při úpravě vyhledávacího řetězce je viewModelu posílána i aktuální pozice kurzoru. Díky tomu je pak možné zobrazit nápovědu s výběrem (obrázek 5.7 vpravo). Tag do něj je zařazen tehdy, pokud aktuálně upravované slovo má totožný prefix s jeho názvem a zároveň již není zahrnut ve vyhledávacím řetězci. Jednotlivé tagy jsou zobrazeny jako buňky ve View, jehož třída je pojmenována jako `TagHelpView`. Ta dědí z `UICollectionView`. Buňky se pak zobrazují podobně jako je tomu při ukládání jízdy, o kterém se píše v kapitole 5.5. Třída zapouzdřuje logiku `UICollectionView`, jeho `dataSource` i `delegate`. Pro komunikaci je pak vytvořen protokol `TagHelpViewDelegate` s jedinou metodou



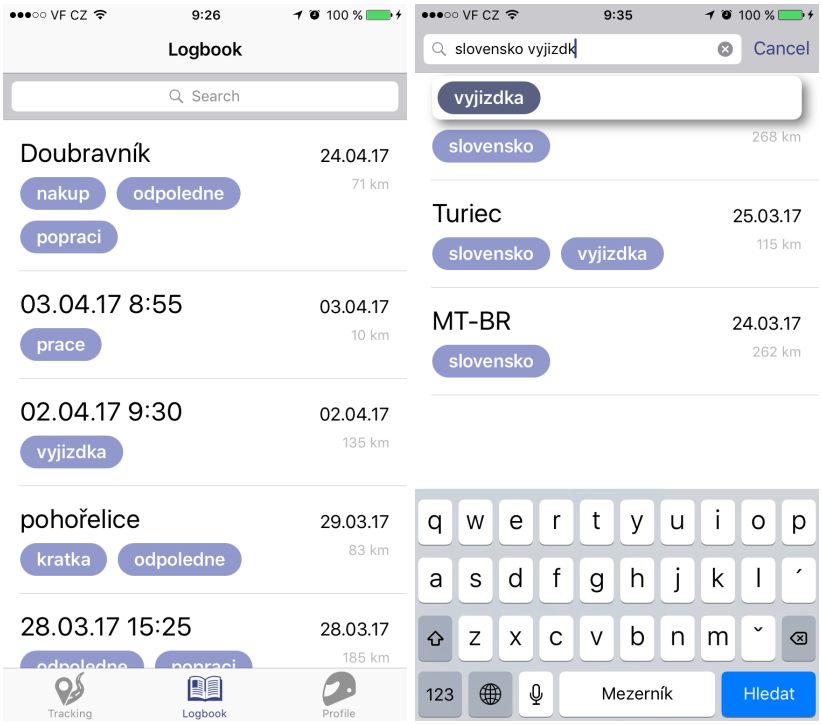
Obrázek 5.6: Vyhledávání jízdy – Vlevo situace, kdy na aktuálním UIViewControlleru, prezentovaném z `UITabBarViewController`, byl nastaven příznak `definesPresentationContext` na hodnotu `false`. To způsobilo posunutí výsledků o výšku `UITabBar`. Vpravo je atribut již nastaven na kladnou hodnotu, což chybu opravilo.

- `func tagHelpView(_ tagHelpView: TagHelpView, didSelectTagAt indexPath: IndexPath)`

Ten je implementován třídou `LogbookResultsViewController`. Metoda je volána při vybrání buňky. Ta předá informaci ViewModelu, který doplní název tagu. Následně se aktualizuje výsledek hledání.

5.7 Detail zaznamenané jízdy

Obrazovka detailu jízdy (obrázek 5.8) je rozdělena do dvou částí, podobně jako tomu bylo u obrazovky sloužící k ukládání jízdy. Horní část obsahuje mapu, spodní část obsahuje údaje o jízdě, přičemž celý obsah je skrolovatelný. Ve spodní části jsou údaje o začátku a konci zaznamenávání, ujetá vzdálenost, celkový strávený čas. Každý údaj je prezentován v `UIView`. V něm se vlevo nachází název a vpravo hodnota. Dále je zde zobrazen graf, jehož jednotlivé sloupce označují orientační rychlost a výška označuje dobu strávenou v dané rychlosti. Vpravo v navigačním panelu je umístěno tlačítko `Edit`. To slouží k editaci záznamu. Po kliknutí je daný ViewController prezentován modálně.



Obrázek 5.7: Historie jízd – **vlevo:** historie seřazená podle data pořízení záznamu, **vpravo:** vyhledávání s nápovědou tagů.

5.7.1 Graf časů strávených v rychlostech

Pro získání hodnot, které se mají v grafu zobrazit, je potřeba nejprve zjistit čas strávený v jednotlivých hodnotách rychlosti. Pro zjednodušení je rychlost v tomto algoritmu diskrétní veličinou, a to od momentu, kdy je uložena do přípravné tabulky. Od té se vyvíjí další výpočty. Tato tabulka je reprezentována typem `Dictionary`; jejím klíčem je rychlost typu `Int` a hodnota typu `NSTimeInterval`, což je alias pro typ `Double`. V rámci každé části záznamu jsou procházeny získané lokace, které obsahují i potřebnou rychlost v momentě průjezdu místem. Ty jsou seřazeny podle času a následně je nad nimi iterováno. Pokud rozdíl rychlosti mezi dvěma po sobě jdoucími lokacemi je maximálně 1, pak polovina rozdílu času mezi nimi je přičtena k času, který byl uložen pro danou rychlost v tabulce. Nová hodnota pak starou nahradí. Pokud podmínka neplatí, je tabulka vyplněna následovně:

- Je vypočítán rozdíl rychlosti a následně krok času

$$\Delta v = v_i - v_{i+1} \tag{5.1}$$

$$t_{step} = \frac{\Delta v}{\Delta t} \tag{5.2}$$

- Pro každou rychlost

$$v_j \in \mathbb{Z}, v_j \in [v_i, v_{i+1}]$$

je prováděn výpočet

$$t_{v_j} = t_{v_j} + t_{step} \tag{5.3}$$

Rychlosti jsou pak shlukovány do intervalů. Jejich počet je získán podílem maximální rychlosti a vhodným počtem sloupců, které se vejdou na obrazovku telefonu. Testováním bylo určeno číslo 10. Při tomto množství je pod sloupci dostatečný prostor pro zobrazení popisu. Pokud by při výpočtu došlo k tomu, že interval je tak malý, že sloupců by bylo více než je ideální počet, pak je získaná velikost intervalu inkrementována. Shlukování pak probíhá v tolika iteracích, kolik má být sloupců. V každé iteraci je získána struktura `SpeedRange` obsahující spodní a horní hranici rychlosti, která má být zahrnuta ve sloupci. Čas strávený v intervalu těchto rychlostí je sečten a následně uložen do pole spolu s daným intervalem.

Komponenta pro vykreslení grafu

Samotný graf je vykreslen s použitím vlastní komponenty, která dědí z `UIView`. Rozložení prvků je vytvořeno pomocí nástroje Interface Builder v souboru xib. To obsahuje několik zanořených `UIStackView`. Samotná komponenta nebyla vytvořena přímým děděním třídy `UIStackView`, neboť se jedná o třídu, která nepodporuje renderování a pouze nastavuje rozmístění prvků, které do ní byly vloženy. Z tohoto důvodu by nebylo možné upravit vzhled komponenty[6].

Jak je vidět na obrazovce detailu jízdy na obrázku 5.8 vpravo, komponenta grafu obsahuje nadpis. Tento text je zobrazen pomocí `UILabel`, který je vložen do `UIStackView`. To má v atributu `axis` nastavenou hodnotu `Vertical` a proporcionální distribuci. V tomto `UIStackView` je obsažen ještě jeden `UILabel`, který se v případě detailu jízdy nepoužívá. Texty je možné konfigurovat pomocí ViewModelu, který implementuje definovaný protokol.

Spodní `UIStackView` obsahuje dva View, které jsou rovněž typu `UIStackView`. Pravý slouží pouze k prezentování maximální a minimální hodnoty osy. Je rozděleno do čtyřech sekcí. První a třetí zobrazuje separátor. Ve druhé se nachází hned dva View typu `UILabel`. Horní je přichycen k vrchnímu okraji a spodní ke spodnímu okraji svého `superview`. Čtvrtá část slouží pouze jako doplněk, aby `subviews` tohoto `UIStackView` byly distribuovány stejně jako sloupce, které se zobrazují v levé části. Sloupec je vytvořen velmi podobným způsobem. Rozdíl je ve druhé části, která obsahuje místo dvou `UILabel` jedno `UIView`, které reprezentuje sloupec. Má definovanou výšku pomocí autolayout. Ta se pak mění podle ViewModelu. Sloupec je dále přichycen k horní a spodní hraně `superview`. Horní však není pevná, ale velikost mezery, které omezení definuje, může být větší nebo rovná konstantě, kterou má nastavenou.

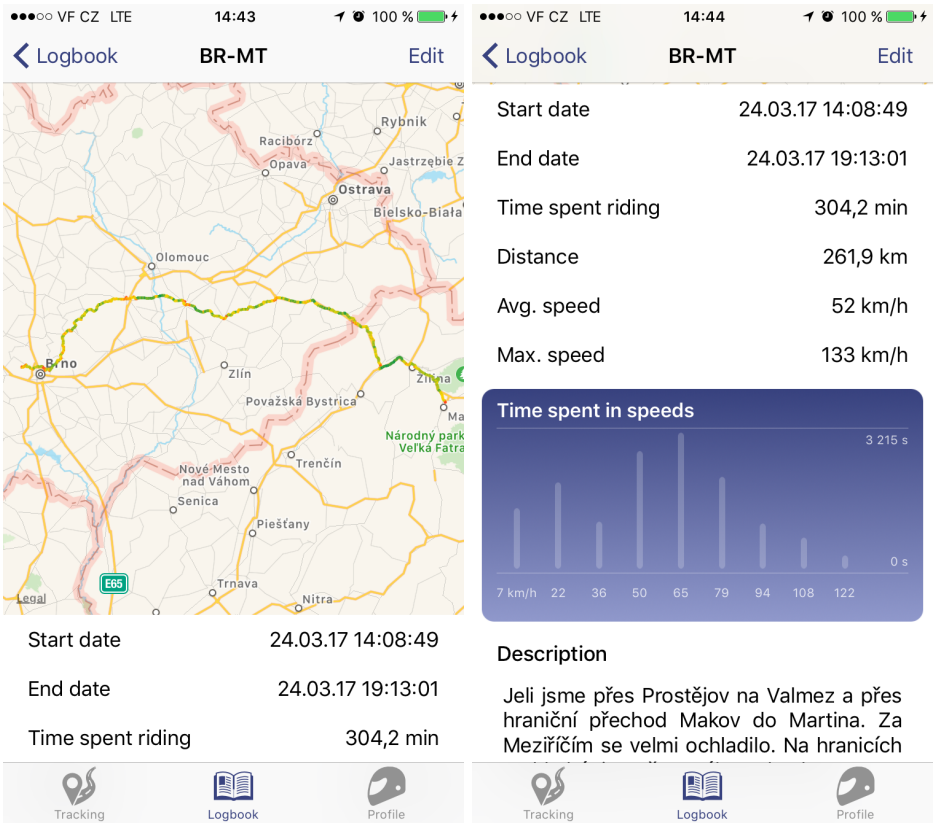
5.7.2 Vykreslení trasy na mapě

Na detailu jízdy je vykreslování trasy náročnější, než je tomu při zaznamenávání jízdy, neboť vyznačené trase se mění barva podle rychlosti průjezdu. Implementaci je možné provést několika způsoby.

Nejjednodušší je se snažit znovu použít kód pro vykreslování trasy z průběhu získávání záznamu. Je zde však rozdíl spočívající v tom, že mezi každými dvěma body musí být zjištěna rychlost, podle níž se pak určí barva ze spektra červená – žlutá – zelená (červená označuje nízkou rychlost, žlutá průměrnou a zelená vysokou).

Pro rychlost menší než je rychlost průměrná, je barva určena postupem:

[6] Zdroj: *Apple Inc.:* API Reference – UIStackView[1] https://developer.apple.com/reference/uikit/uistackview



Obrázek 5.8: Detail zaznamenané jízdy z Brna do Martina. Obrazovka obsahuje mapu s vykreslenou trasou, která je barevně odlišena podle rychlosti průjezdu v daném místě. V spodní části jsou informace o jízdě včetně grafu časů strávených v rychlostech. Ten je zobrazen pomocí vytvořené komponenty

.

- Je vypočten poměr rychlosti k průměrné rychlosti

$$\lambda = \frac{v}{\overline{v}} \tag{5.4}$$

- Dále určíme jednotlivé kanály barvy, přičemž c je vektor barevného modelu RGB

$$\mathbf{c} = \underline{\mathbf{c}} + \lambda\overline{\mathbf{c}} \tag{5.5}$$

Pro rychlost větší nebo rovnou průměrné rychlosti výpočet provedeme následovně:

- Vypočítáme poměr

$$\lambda = \frac{v - \overline{v}}{v_{max} - \overline{v}} \tag{5.6}$$

- Dále každý kanál barvy:

$$\mathbf{c} = \overline{\mathbf{c}} + \lambda\hat{\mathbf{c}} \tag{5.7}$$

Barva bude uložena jako atribut nově vytvořené třídy dědící z `MKPolyline`. Podobně jako v kapitole o záznamenávání jízdy se instance této třídy přidají do mapy pomocí metody `addOverlays`. `UIViewController` této obrazovky rovněž musí implementovat protokol `MKMapViewDelegate` s metodou `func mapView(_ mapView: MKMapView, rendererFor`

`overlay: MKOverlay) -> MKOverlayRenderer`. Jak bylo zjištěno během testovaní, zobrazení obrazovky funguje bez potíží, pokud je záznam dlouhý v řádu několika minut. V případě tříhodinového záznamu, během kterého bylo zaznamenáno 8000 změn polohy, se doba načtení obrazovky značně prodloužila, a to přibližně na 18 sekund. Během této doby je zablokováno vlákno uživatelského rozhraní, což způsobuje zamrznutí aplikace.

Jedním z řešení tohoto problému je minimalizovat počet částí vzhledem k měřítku a zároveň přidávat části trasy postupně, podle aktuálně zobrazené části mapy. Tento způsob by byl však komplikovaný vzhledem k problematickému určování, který bod může být vyloučen z vykreslování tak, aby nedošlo k přílišným vadám vykreslování trasy v obloucích s malým poloměrem.

Dalším řešením je implementace vlastního `MKOverlayPathRenderer`, který má za úkol rozhodnout, co a jak se má vykreslit. To se děje podle dat, která mu jsou předložena instancí třídy `ColoredPolylineOverlay` implementující protokol `MKOverlay`. Ta byla v rámci práce vytvořena. Její inicializátor obsahuje parametr pro předání pole se strukturami `CLLocation`. Tato struktura je zvolena, protože obsahuje informace o pozici i rychlosti v momentě průjezdu místem. Prvky pole jsou inicializovány z objektů třídy `Location`, které jsou k jízdě asociovány. Dalším parametrem je maximální a průměrná rychlost v celém záznamu, aby bylo možné získat korektní barvu znázorňující rychlost mezi dvěma lokacemi, jak bylo popsáno výše. Pro splnění požadavků protokolu `MKOverlay` třída obsahuje atributy:

- `var boundingMapRect: MKMapRect` – obdelník definuje minimální plochu, která obsahuje všechny body, mezi kterými má být trasa vykreslena
- `coordinate: CLLocationCoordinate2D` – obsahuje geografický střed obdelníku definovaný v `boundingMapRect`

Dále má třída atribut, do něhož se ukládá pole struktur `Point`. Každá z nich obsahuje bod na mapě `MKMapPoint` a strukturu nesoucí hodnoty kanálů RGB určující barvu linky, která povede přes tento bod. `MKMapPoint` obsahuje souřadnice, která byla navržena pro použití v dvourozměrné mapě. Vždy odkazuje na místo v reálném světě a lze je převést na `CLLocationCoordinate2D`, což je struktura uchovávající geografické souřadnice. [7] Ta je například typem atributu `coordinate` ve struktuře `CLLocation`.

Vlastní třída dědící z `MKOverlayPathRenderer` má inicializátor s parametrem typu `ColoredPolylineOverlay`. Ten si uloží, a dále s ním budou pracovat metody, které přepisují implementaci ze své rodičovské třídy. Těmi jsou:

- `func createPath()`
- `func draw(_ mapRect: MKMapRect, zoomScale: MKZoomScale, in context: CGContext)`

První zmíněná metoda by se měla přepisovat, neboť slouží k vytvoření `CGPath` a uložení do atributu `path`. Ten se může použít ke zjištění, zda-li je křivka obsažena v právě zobrazeném výseku mapy ve druhé zmíněné metodě. Hodnota je získána tak, že se vytvoří `CGMutablePath`. Nad ním je zavolána metoda `move(to point: CGPoint)`, jejíž parametrem bude hodnota zkonvertovaná z atributu `mapPoint` prvního prvku pole struktur `Point`. Ta je vlastněna overlayem, s kterým bude instance této třídy inicializovaná. Další prvky

[7] Zdroj: *Apple Inc.:* API Reference – CLLocationCoordinate2D[1] https://developer.apple.com/reference/corelocation/cllocationcoordinate2d

zmíněného pole budou též zkonvertovány na `CGPoint`, ale ty se použijí jako parametr metody `addLine(to point: CGPoint`, která se zavolá nad vytvořenou `path`.

Druhá metoda slouží k samotnému vykreslování. Následujícím postupem se zjistí, zda-li je `path` obsažena ve výřezu mapy, který je prvním parametrem metody.

1. Daný výřez mapy z parametru `MKMapRect` je zkonvertován na `CGRect` pomocí metody `func rect(for mapRect: MKMapRect) -> CGRect`, která je implementována rodičovskou třídou.

2. Zkonvertovaný výřez se použije jako parametr metody `intersects(_ rect2: CGRect)`, která se zavolá nad atributem `boundingBox`, který je vlastněn proměnnou `path`.

Pokud by návratovou hodnotou poslední volané funkce bylo `false`, nebude se nic vykreslovat. V opačném případě se nad všemi všemi body uloženými v `ColoredPolylineOverlay`, kromě prvního, vykonají následující kroky:

1. Je vytvořena `CGMutablePath`.

2. Je nastavena její výchozí pozice pomocí metody `move(to point: CGPoint`, jejíž parametrem je předchozí bod `ColoredPolylineOverlay`.

3. Je přidána čára do `CGMutablePath` pomocí metody `addLine(to point: CGPoint)`.

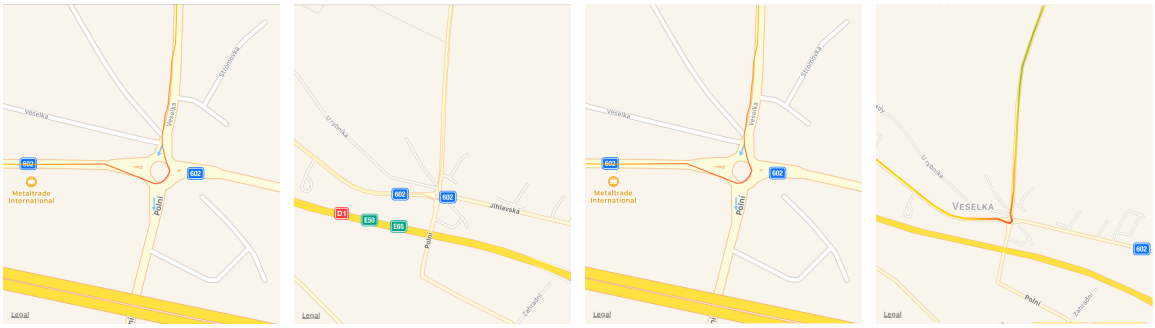
4. Je zjištěna šířka čáry návratové hodnoty metody `convertToUserSpace(_ size: CGSize) -> CGSize`. Ta je zavolána nad instancí tříy `CGContext`, která je parametrem metody určené k vykreslování. Parametrem volání je velikost čtverce, jehož stranou je atribut `lineWidth` třídy `MKOverlayPathRenderer`. Metoda je použita, aby čáry byly vždy vykreslené přibližně stejnou šířkou nezávislou na přiblížení mapy. Na obrázku 5.9 vpravo je vidět vykreslení při použití tohoto mechanismu k získání podobné šířky čáry za jakéhokoliv zvětšení mapy. Vlevo je pak zobrazeno vykreslení při použití konstantní šířky.

5. Nad vytvořenou `path` je zavolána metoda `copy(strokingWithWidth lineWidth: CGFloat, lineCap: CGLineCap, lineJoin: CGLineJoin, miterLimit: CGFloat) -> CGPath`, přičemž všechny parametry, kromě `lineWidth`, obsahují hodnoty stejnojmenných atributů `MKOverlayPathRenderer`. Parametr `lineWidth` obsahuje hodnotu šířky získanou v předchozím kroku.

6. V tento moment se začíná manipulovat s nastavením `CGContext`. Ten obsahuje stav, který je uložen zavoláním metody `saveGState()` nad jeho instancí. Tento krok by se měl provést, neboť se bude měnit nastavení kontextu [8]. Vytvořená kopie `CGPath`, která již obsahuje požadované nastavení, je vložena do aktuálního `CGContext` pomocí metody `addPath(_ path: CGPath)`.

7. Nad `CGContext` je zavolána metoda `clip`, která zajistí, že změny na kontextu budou aplikovány pouze na přidanou `path`.

8. Nyní se již bude vykreslovat čára pomocí metody `func drawLinearGradient(_ gradient: CGGradient, start startPoint: CGPoint, end endPoint: CGPoint, options: CGGradientDrawingOptions)`, přičemž `startPoint` a `endPoint` jsou ty stejné body, které byly použity pro nastavení `CGMutablePath`.



Obrázek 5.9: Vykreslování trasy při různém přiblížení mapy – Použití pevné šířky čáry způsobí při velkém oddálení mapy neviditelnost trasy, jak je zřejmé na obrázcích vlevo. API poskytuje metodu k získání velikosti vykresleného obdélníku vzhledem k přiblížení mapy. Jedná se o metodu `convertToUserSpace(_ size: CGSize) -> CGSize`, kterou implementuje třída `CGContext`. Pomocí ní lze získat velikost obdélníku pro nastavení mapy. Díky této metodě je možné získat i šířku čáry tak, aby byla vždy požadovaně silná.

Parametr gradient je naplněn objektem požadovaného typu. K jeho inicializaci stačí pouze barvy, které mají být použity při vykreslování efektu.

9. Nyní je zavolána metoda `restoreGState`, která nahradí poslední uložený stav aktuálním stavem.[8]

Instance této třídy se vytvoří v delegátu mapy v metodě `func mapView(_ mapView: MKMapView, rendererFor overlay: MKOverlay) -> MKOverlayRenderer`, a to v případě, že `overlay` bude typu `ColoredPolylineOverlay`.

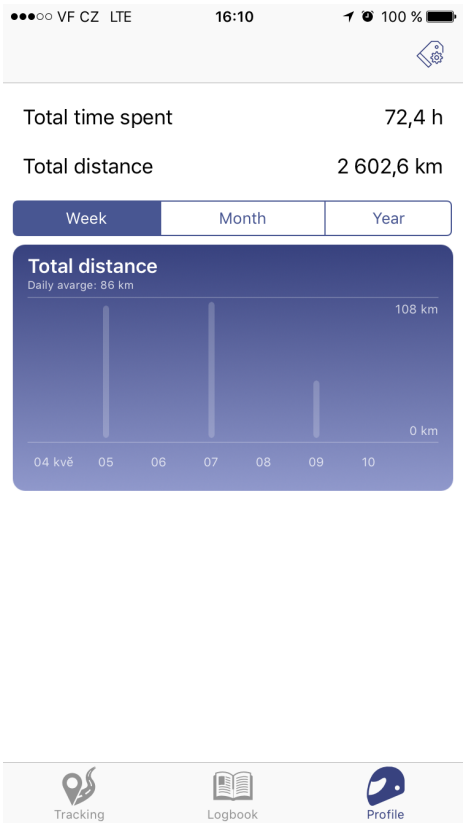
Toto řešení umožňuje vykreslení trasy bez velkých prodlev během změny přiblížení a pohybu s mapou přibližně do 100 km délky záznamu. Pak nastávají prodlevy během překreslování trasy při pohybu mapou a při změnách jejího přiblížení. To způsobuje snaha rendereru o vykreslení úseků, které jsou mimo aktuálně zobrazovanou část mapy, neboť obdélník definující oblast, ve kterém má být trasa vykreslena je příliš velký. Z tohoto důvodu pak renderer mylně určí, že je potřeba trasu vykreslit. Řešením je záznam rozdělit do více částí, přičemž každá obsahuje maximálně 200 zaznamenaných bodů. Díky tomu je obdélník, zahrnující jednotlivé body trasy, dostatečně malý na to, aby se renderer dokázal správně rozhodnout, jestli má být daná trasa vykreslena.

5.8 Statistiky uživatele

Na obrázku 5.10 je obrazovka se statistikami uživatele. Ta je vytvořena pomocí `UIStackView`, které obsahuje 3 prvky. Prvním prvkem je `UIStackView`, v němž jsou vloženy dvě `UIView`. Každé z nich obsahuje dvě View typu `UILabel`. Ty slouží pro zobrazení hodnoty celkově stráveného času řízením(v případě použití automatické pauzy) a celkové zaznamenané vzdálenosti. Druhý a třetí prvek jsou k sobě vázány. V třetím je zobrazen graf ujetých kilometrů za poslední týden, měsíc nebo rok. Interval je určen zvolením volby v `UISegmentControl`, který je druhým prvkem. Z obrazovky lze kliknutím na tlačítko v navigačním panelu otevřít seznam tagů, které se v aplikaci nachází.

Graf je zobrazen pomocí komponenty, jejíž tvorba je popsána v kapitole 5.7.1. Ten je naplněn ViewModelem, který připraví data k zobrazení. Při jeho instanciaci je vytvořena

tabulka ujetých vzdáleností za poslední rok. Z této tabulky jsou pak získávány data pro vybraná období.



Obrázek 5.10: Obrazovka zobrazuje statistiky uživatele mezi kterými je i graf ujetých kilometrů za poslední týden, měsíc nebo rok. Pomocí přepínání hodnot se uživatel dozví i průměrný denní nájezd kilometrů ze zvoleného období.

Kapitola 6

Testování a další pokračování projektu

V této kapitole je popsáno testování aplikace. Jednalo se o testování funkcionality v reálném provozu a o uživatelské testování. Na základě získaných poznatků a nápadů je navržen další vývoj projektu.

6.1 Testování funkčnosti

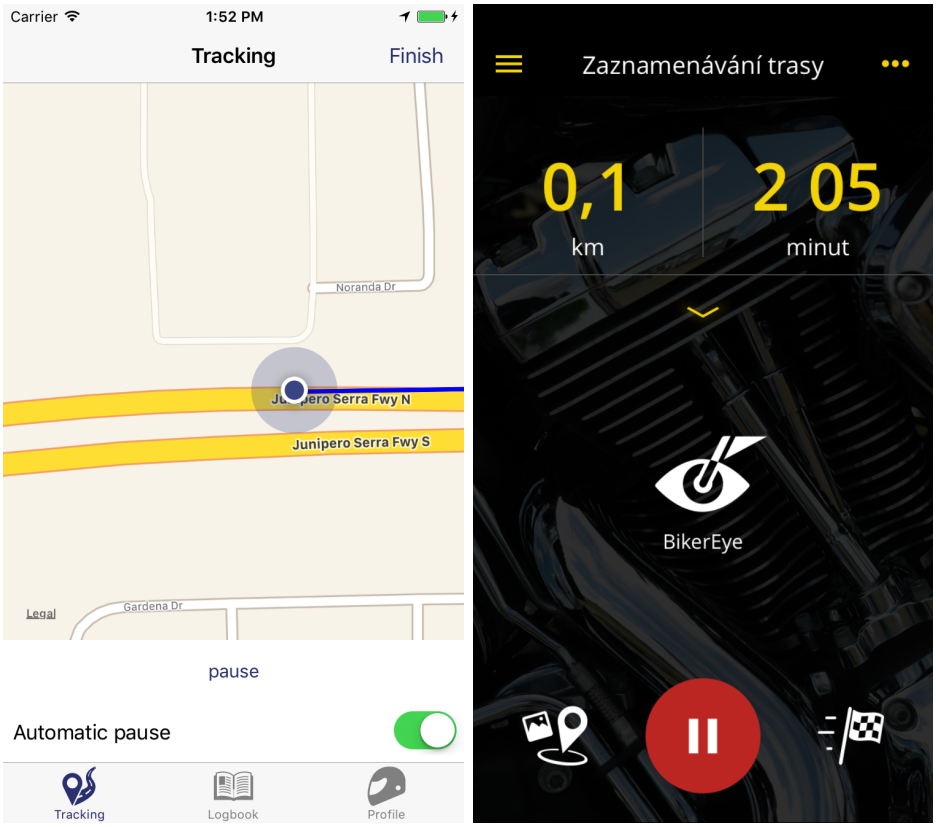
Od prvních verzí aplikace bylo prováděno testování v reálném provozu, které bylo umožněno díky vhodnému naplánování práce během jednotlivých iterací vývoje. To vedlo k odhalení několika problémů s nadměrným využíváním prostředků systému, které tak mohly být hned opraveny.

První funkcionalitou bylo zaznamenávání trasy, ke které se následně přidala historie jízd. Z ní je přístupný detail záznamu, který obsahuje mapu s vyznačenou trasou. Její části jsou barevně odlišeny podle rychlosti, která byla v daném úseku zaznamenána. V původní implementaci vykreslení trasy se při zobrazení detailu s velmi dlouhým záznamem značně zpomalilo načtení obrazovky. Způsob vyřešení tohoto problému byl popsán v kapitole 5.7.2.

V druhé polovině vývoje byla řešena automatická pauza jako poslední důležitá funkcionalita z pohledu vylepšení, bez níž se však aplikace obejde. Ta byla testována v hustém provozu i v tunelech. Během jízdy na motocyklu se ukázala jako funkční, i když je zde potenciál pro chybné detekování pauzy v případě jízdy v tunelech.

6.2 Uživatelské testování

Uživatelské testování aplikace bylo provedeno na pěti lidech bez znalosti problematiky vývoje aplikací. Dva z nich ji alespoň jednou použili v reálném provozu. Při úkolech, které jim byly zadány, jsem sledoval jejich počínání. Tyto úkoly následně byly jimi provedeny i v konkurenční aplikaci BikerSeason, která byla od spuštění v nevýhodě kvůli registraci, kterou uživatelé museli vykonat, aby ji mohli vůbec použít. Ta v mém řešení není potřeba, neboť se zatím žádné informace neukládají na vzdálený server, což testeři vnímali jako výhodu.



Obrázek 6.1: Srovnání vytvořené obrazovky (vlevo) pro zaznamenávání s obrazovkou zaznamenávání v aplikaci BikerSeason(vpravo) – Neznalému uživateli trvalo čtyřikrát delší dobu uložení trasy v BikerSeason, než tomu bylo v mé aplikaci. Vliv na delší dobu ukládání měla tlačítka, jejichž funkce je popsána ikonami, které byly špatně interpretovány.

Zaznamenání jízdy

Uživatelé měli za úkol zaznamenat a uložit krátkou jízdu dle možností aplikace. V BikerSeason trvalo ukončení a uložení záznamu přibližně třikrát déle než při používání mé aplikace. Prodleva v Bikerseason nastala hlavně při snaze o ukončení zaznamenávání, kdy uživatel nedokázal rozpoznat správnou funkci tlačítek popsanými ikonkami na obrazovce. V mé aplikaci jsou tlačítka popsány textem. Z toho můžeme usoudit, že zejména v počátcích používání aplikace je text pro předání informace o funkci tlačítka lepší než ikona, která může být chybně interpretovaná.

Následné uložení přibližně proběhlo stejně rychle v obou aplikacích, přičemž v mém řešení byly přiřazeny i tagy. Několik jich bylo v aplikaci předdefinováno.

Vyhledání záznamů

Mezi hlavní funkcionalitu, která měla být největší inovací v aplikacích pro zaznamenávání trasy jízdy na motocyklu, byla organizace jízd. Testování bylo na toto zaměřeno. Uživatelé měli za úkol zjistit počet všech jízd, které byly alespoň částečně zaznamenány na Slovensku. V obou aplikacích je dostupné hledání, proto by se dal předpokládat podobně rychlý výsledek. V aplikaci BikerSeason uživatelé téměř okamžitě klikli na ikonu filtru, kde našli vyhledávací pole a následně do něj zadali řetězec „Slovensko“. K výsledku se tímto způso-

bem však nedostali. Po nějaké době zjistili, že výsledek sice obsahuje trasy na Slovensku, ale nejsou vytvořeny k aktuálně přihlášenému účtu. Pak zvolili ve filtru kategorii „Moje trasy“ vyhledávat manuálně. Dva testeři se nedostali ke správnému výsledku, neboť přehlédli záznam, který splňoval podmínku ze zadání. V mé aplikaci byly všechny zahraniční jízdy opatřeny tagem, který označoval, v jaké zemí byl záznam pořízen, až na jednu, kde namísto tagu byl použit popisek. Testeři po vstupu do historie jízd viděli vyhledávací pole a rovnou se ho pokusili použít. Po napsání prvních znaků z názvu státu se zobrazila nabídka s nápovědou tagů, kterou hbitě využili a vyhledání všech záznamů bylo téměř instantní.

Dalším úkolem týkajícím se vyhledávání bylo nalezení jízdy z konkrétního data. Tento úkon se v BikerSeason musel opět provádět manuálně, tedy prostým procházením seznamu. Dva testeři v mé aplikaci zkusili zadat datum do vyhledávacího pole. To obsahovalo chyby ve formátu, na které je *data detektor* z knihovny Foundation velmi náchylný, což zapříčinilo nezjištění data v řetězci a následné nevyhledání výsledků. Tři testeři zvolili způsob hledání manuální cestou, čili skrolováním skrz seznam jízd a jejich výsledek byl srovnatelně rychlý v obou aplikacích. Vzhledem k aktuální funkčnosti detektoru je pro vyhledání jízd podle data vhodnější vytvořit filtr, ve kterém bude možné definovat interval období, pro které se mají výsledky hledání zobrazit.

Z uživatelského testování lze usoudit, že přidávání tagů je vhodným řešením k organizaci jízd, který zároveň pomůže k vyhledávání jízd, za podmínky, že jsou záznamy poctivě ukládány s relevantními informacemi.

Celkové orientování se v mém řešení se jevilo oproti konkurenci, která používá „Hamburger menu“, jako rychlejší. To přisuzuji použití nativních řešení pro průchod aplikací. Těmi je lišta se záložkami a navigační panel.

6.3 Další vývoj projektu

Na základě zjištěných informací z testování bude do aplikace přidána možnost filtrování jízd podle data. Dále by měla být přidána možnost přidání a upravení přestávek v zaznamenané jízdě. Tuto funkci by mohli využít lidé, kteří automatickou detekci přestávek nechtějí z nějakého důvodu využít. Mezi ně může patřit například jízda mimo silnice, která je od ní velmi odlišná v závislosti na povaze terénu. Dále by aplikace mohla být rozšířena o mapu, na které by byly vyznačeny navštívené oblasti.

Další vývoj projektu by měl také pomoci k rozšíření aplikace v komunitě motorkářů. K tomu je velmi důležité umožnit uživateli předat informace o zaznamenaných jízdách dalším lidem. To by mohlo být provedeno pomocí vytvořené webové stránky, která by s mobilní aplikací používala jeden server. S tím by do aplikace přibyla možnost dobrovolné registrace a možnost ukládat záznamy na vzdálené uložiště. V případě, že by se uživatel nechtěl registrovat, možnost sdílení trasy by měla být zachována. Představa je taková, že požadovaný záznam by byl odeslán na server, který by vygeneroval webovou stránku s URL, kterou by uživatel následně sdílel.

Kapitola 7

Závěr

Cílem práce bylo seznámení se s problematikou tvorby aplikací pro iOS a vytvoření funkční aplikace pro zaznamenávání jízdy na motocyklu. Byly popsány existující nástroje pro záznam trasy. Na základě získaných poznatků byla navržena funkčnost aplikace pro platformu iOS.

Díky získaným informacím se povedlo vytvořit konkurence schopný produkt, téměř připravený k distribuci. Aplikace obsahuje řadu inovací na poli aplikací pro zaznamenávání trasy jízdy na motocyklu. Mezi ně patří zejména vylepšená automatická pauza v porovnání s konkurenčními aplikacemi a dále správa jízd, která ulehčila vyhledávání konkrétního záznamu. Uživatelé se více dozví o průběhu své jízdy. Jedním z prostředků je graf popisující, jakou dobu řidič strávil v rychlostech naměřených během pořizování záznamu. Další informace lze vyčíst z trasy zobrazené na mapě. Její části jsou barevně odlišeny podle rychlosti, kterou byl úsek projížděn. Velkou výhodou aplikace je možnost zaznamenávání jízdy zcela bez nutnosti registrace a připojení k internetu. To využijí hlavně uživatelé cestující po světě. Práce dále obsahuje návrh dalších možností rozšíření vytvořeného produktu.

Literatura

[1] Apple Inc.: *API Reference*. [Online; navštíveno 18.4.2017].
URL https://developer.apple.com/reference/

[2] Apple Inc.: *Guides and Sample Code*. [Online; navštíveno 18.4.2017].
URL https://developer.apple.com/library/content/navigation/

[3] Apple Inc.: *Swift Package Manager Project*. [Online; navštíveno 17.4.2017].
URL https://github.com/apple/swift-package-manager

[4] Apple Inc. and the Swift project authors: *Advanced App Capabilities*. [Online; navštíveno 17.4.2017].
URL https://developer.apple.com/support/app-capabilities/

[5] DeAmicis, M.: *BMW Motorrad Partners With Rever*. Červenec 2016, [Online; navštíveno 3.4.2017].
URL http://www.bmwblog.com/2016/07/18/bmw-motorrad-partners-rever/

[6] Furrow, A.: Introduction to MVVM. [Online; navštíveno 18.4.2017].
URL https://www.objc.io/issues/13-architecture/mvvm/

[7] Hanning, T.: *Data Persistence On iOS – An Overview*. [Online; navštíveno 18.4.2017].
URL http://www.thomashanning.com/data-persistence-ios/

[8] Neuburg, M.: *Programming iOS 9: Dive Deep into Views, View Controllers, and Frameworks*. O'Reilly Media, December 2015, ISBN 978-1-491-93685-6.

[9] Orozco, G.: *MVVM and Coordinator pattern together*. [Online; navštíveno 18.4.2017].
URL https://medium.com/@giovannyorozco24/mvvm-and-coordinator-pattern-together-8920fc0f1f55

[10] *Objective C*. [Online; navštíveno 17.4.2017].
URL https://cs.wikipedia.org/wiki/Objective-C

[11] *Povinná výbava v Evropě*. [Online; navštíveno 31.3.2017].
URL http://www.uamk.cz/garaz/1333-povinna-vybava-v-evrope

Přílohy

Příloha A

Obsah přiloženého paměťového média

- Projekt.zip – archiv obsahuje zdrojový kód aplikace
- Plakat.pdf
- Video.mp4
- Design – složka obsahuje návrh aplikace vytvořený v nástroji Interface Builder